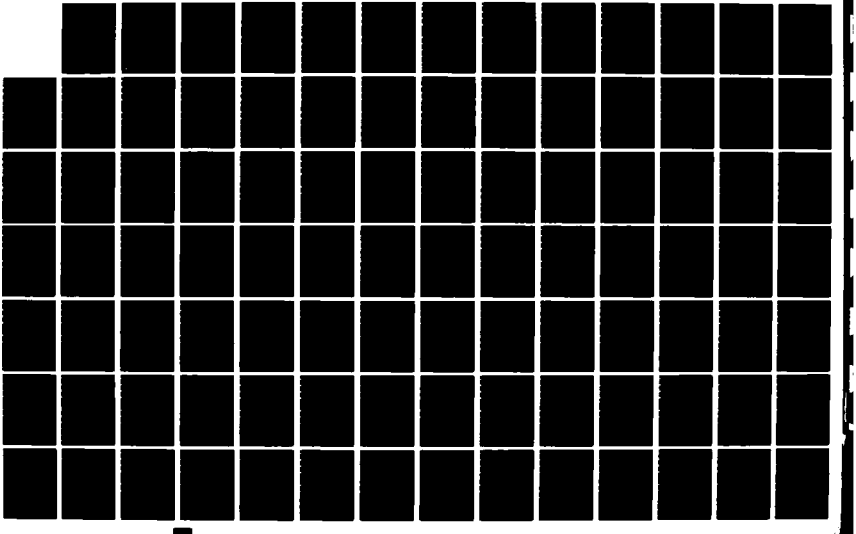
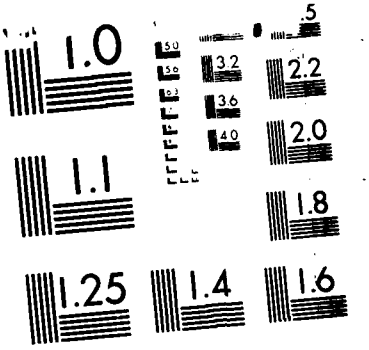
AD-A166 373 A SINGLE-PHASE METHOD FOR QUADRATIC PROGRAMMING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH S C HOYLE NOV 85 AFIT/CI/NR-86-3D 1/2

UNCLASSIFIED F/G 9/2 NL





SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 86- 3D	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Single-Phase Method for Quadratic Programming		5. TYPE OF REPORT & PERIOD COVERED ~~THESIS~~/DISSERTATION
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen Carey Hoyle		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: Stanford University		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433-6583		12. REPORT DATE ~~1986~~ Nov 85 13. NUMBER OF PAGES 110
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASS 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, If different from Report)*

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

LYNN E. WOLAVER 4 Apr 86
Dean for Research and
Professional Development
AFIT/NR, WPAFB OH 45433-6583

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

DTIC ELECTE APR 1 0 1986 S D E

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

AD-A166 373

DTIC FILE COPY

86 4 8 087

A SINGLE-PHASE METHOD FOR QUADRATIC PROGRAMMING

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF OPERATIONS RESEARCH

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

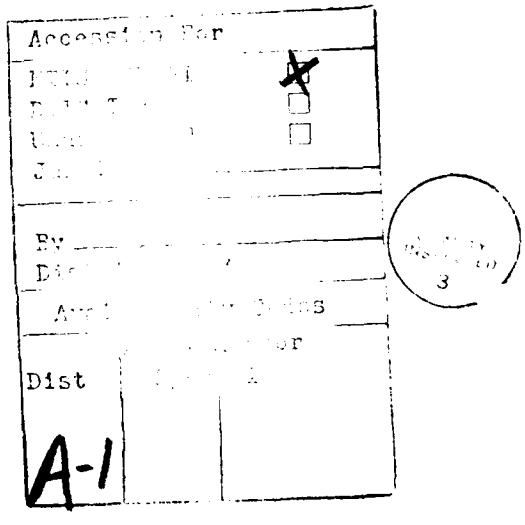
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Stephen Carey Hoyle

November 1985



Abstract

A SINGLE-PHASE METHOD FOR QUADRATIC PROGRAMMING

Stephen Carey Hoyle, Ph.D.
Stanford University, 1986

This thesis will describe a single-phase quadratic programming (QP) method. A need for such a method is apparent when solving a sequence of closely-related QP's. One example of this is in the area of nonlinear programming, which uses sequential quadratic programming. Another example is sensitivity analysis involving the constraints. In both of these examples, if the quadratic problems are large-scale and the initial point is infeasible, the single-phase method of this thesis may be particularly useful.

If the Hessian is indefinite, the method will find a local minimum when solving a dense QP, and a constrained stationary point when solving a large-scale QP. If the Hessian is positive definite, it will find the global minimum. Additionally, if the QP is known to be positive definite, the method converges more quickly than if the QP were treated as a general indefinite problem.

The single-phase method of this thesis is an active-set method which solves a sequence of equality-constraint quadratic programs (EQP). It differs from other active-set methods in that the current iterate may *violate* constraints in the working set (the prediction of the active set). Special care has been taken to avoid rank deficiency in the working set. An efficient means of solving the successive EQP's of

Acknowledgements

I would like to thank Dr. Philip E. Gill and Prof. Walter Murray for their guidance and encouragement throughout my research. I am indebted to the US Air Force for allowing me to pursue this degree and for providing financial assistance.

The constant support from my wife, Claudia, in the years before coming to Stanford and during the years here has been very important to me. I would like to dedicate this dissertation to her and to our son, Andrew, who has brought so much joy into our lives since his birth this year.

a large-scale problem is included.

Finally, an implementation of the method for dense QP's is described. The implementation includes several new strategies for deleting constraints from the working set. Computational results of the implementation are discussed. In particular, when solving positive-definite QP's in which the dimension of the null space at the solution was about one half the number of variables, the implementation required approximately one half as many EQP solves as did an efficient two-phase algorithm.

Bibliography

Conn, A. R. and Gould N. I. M. (1984). On the location of directions of infinite descent for nonlinear programming algorithms, *SIAM J. Numer. Anal.* **21**, pp. 1162–1179.

Cottle, R. W. (1974). Manifestations of the Schur complement, *Linear Algebra and its Applics.* **8**, pp. 189–211.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1981). QP-based methods for large-scale nonlinearly constrained optimization, Report SOL 81-1, Department of Operations Research, Stanford University, California. To appear in *Nonlinear Programming 4*, (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), Academic Press, London and New York.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1983a). User's guide for SOL/QPSOL: A Fortran package for quadratic programming, Report SOL 83-7, Department of Operations Research, Stanford University, California.

TABLE OF CONTENTS

List of Tables

Chapter One

Introduction

This thesis will present and describe the implementation of a *single-phase* quadratic programming (QP) method. A single-phase method works towards satisfying optimality and feasibility conditions simultaneously. Many QP methods are *two-phase*, since they first find a feasible point, and then an optimal point. Each QP method must ascertain if the Kuhn-Tucker conditions are satisfied. This can be determined by solving the *augmented system* of linear equations, which state the Kuhn-Tucker optimality and feasibility conditions. If a method solves the augmented system without transformation, then it is called a *Lagrangian method* (see Gould, 1984). If a method transforms the augmented system into several smaller systems, then it is called a *projection method*.

When the QP problem is *dense*, there may be no overwhelming reason to prefer a single-phase method to a two-phase method, or a Lagrangian method to a projection method. This is not true in a *large-scale* problem. There is a fundamental objection to using a projection method on a large-scale problem. The smaller linear systems in a projection method involve forming products of sparse matrices. Products of sparse matrices are generally not sparse. Thus, the projection methods may be computationally expensive since they cannot use sparse matrix techniques. Lagrangian methods do not destroy the inherent sparseness of the problem, since the augmented system is solved without transformation.

There are several objections to using a two-phase Lagrangian method on a large-scale problem. First, different code is usually required for each of the two phases. Second, the factorizations calculated in the first phase cannot usually be

used in the second phase. The work to compute these factorizations is typically a large portion of the total computational effort. Third, a two-phase (projection or Lagrangian) method ignores the objective function during the first phase. This implies that even though the initial infeasible point may be "near" optimal, the first phase may calculate a feasible point that is "far" from optimal. In a single-phase Lagrangian method, the first two objections do not apply. By working towards optimality and feasibility simultaneously, a single-phase method should diminish the third objection. Thus, a single-phase Lagrangian method may be computationally preferable when solving a large-scale QP problem. The purpose of this thesis is to present a single-phase method which can use a Lagrangian method to solve the augmented system.

There are two main reasons for developing this method. First, its computational attractiveness alone makes it worthwhile to study as a different QP method. Second, a need for such a method is apparent when solving a sequence of closely-related QP's. One example of this is in the area of nonlinear programming (NLP), which uses sequential quadratic programming (SQP). Another example is sensitivity analysis involving the constraints. In both of these examples, if the quadratic problems are large-scale and the initial point is infeasible, the single-phase method of this thesis may be particularly useful.

Some QP methods will accommodate an *indefinite* Hessian. Others restrict themselves to the subset of quadratic problems with positive definite Hessians. If the Hessian is indefinite, the single-phase method of this thesis will find a local minimum when solving a dense QP, and a constrained stationary point when solving a large-scale QP. If the Hessian is positive definite, it will find the global minimum. Additionally, if the QP is known to be positive definite, the method converges more quickly than if the QP were treated as a general indefinite problem.

Rank deficiency in the working set is not a theoretical problem in most QP methods. However, because the single-phase method presented here defines the working set differently, rank deficiency is possible. Thus, the management of this rank deficiency is an important part of the single-phase method.

Chapter Two describes the basic steps of an active-set method. It concludes with a description of several alternative active-set QP methods.

Chapter Three presents the single-phase active-set method of this thesis. Before presenting the method, it describes a general strategy to maintain nonsingularity in the augmented system. Also discussed is an efficient method for solving the successive augmented systems of a large-scale QP problem.

In Chapter Four, a single-phase method for dense QP's (*QPSFA*) is described. (*QPSFA* is derived phonetically from single-phase algorithm for QP's) The description explains how the required factorizations are updated, and how the method handles an indefinite Hessian. Also described are various constraint deletion strategies to be tested. The last section compares *QPSFA* to the alternative QP algorithms described in Chapter Two.

Chapter Five presents computational results of the implementation of *QPSFA*. It begins with a description of how the quadratic programs are randomly generated. Then the different constraint deletion strategies of *QPSFA* are compared. Finally, the implementation is compared to an alternative QP algorithm.

Chapter Two

Active-Set Quadratic Programming

2.1. Overview of an active-set method

The quadratic programming problem is assumed to be stated in the following form:

$$\text{QP1} \qquad \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad c^T x + \tfrac{1}{2} x^T H x$$

$$\text{subject to} \quad Ax = b, \quad x \geq 0,$$

where c is a constant n-vector, H is a constant $n \times n$ symmetric matrix, and A is an $m \times n$ matrix. The constraints involving A will be called *general* constraints; the remaining constraints will be called *bounds*. The vector $\hat{g} \equiv c + Hx$ will denote the *gradient* of the quadratic objective function. The vector $r \equiv Ax - b$ will denote the *residual* of the general constraints.

The method presented in this thesis is an *active-set method*. The active set is the set of constraints that are satisfied exactly at the solution. Briefly, an active-set method is an iterative process. At the start of each iteration, the method maintains a prediction of the active set. This prediction is called the *working set*. The method determines if the prediction is correct. If it is correct, the method concludes successfully. If it is not correct, a *search direction* is constructed and a *step length* along the search direction is calculated.

The constraints in QP1 can be written in other equivalent forms. The form used here was chosen for several reasons. First, the general constraints are always in the working set, so changes to the working set will consist only of adding and deleting bounds. This will make the description of the method simpler. Second,

since only bounds in the working set can be changed, we shall primarily be working with the columns of A. In a large-scale problem, where many columns of A are columns of the identity matrix and as such are not stored explicitly, it is easier to work with the columns than the rows of the constraint matrix.

The search direction is constructed so that the general constraints are exactly satisfied at a step length of one, and the bounds in the working set are always satisfied. For a bound constraint in the working set, this construction is achieved by setting the corresponding component of the search direction to zero. Thus, the associated variable is *fixed*, and specification of the working set induces a partition of x into *fixed* and *free* variables. Let the subscripts "$_{FR}$" and "$_{FX}$" denote a vector or matrix whose elements are associated with the free and fixed variables, respectively. During a given iteration, the fixed variables are effectively removed from the problem. Hence, we let p denote the search direction *with respect to the free variables only*.

Let α denote the step length. Once the search direction and step length are specified, the new iterate, $\bar{x}$, is defined by

$$\bar{x}_{FR} = x_{FR} + \alpha p.$$

Due to the quadratic nature of the objective function, there are only two choices of step length. A step of unity along p is the exact step to the minimum of the quadratic objective function restricted to the working set. If a step of unity can be taken, the next iterate will be a constrained stationary point with respect to the working set, and exact Lagrange multipliers can be computed to determine if a constraint should be deleted. Otherwise, the step along p to the nearest constraint is less than unity, and a new constraint will be included in the working set at the next iterate. Thus, the working set is *updated* by possibly adding or deleting a

constraint before the next iteration begins. For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.

Calculation of the search direction will now be considered. Let H and g denote the components of $\mathcal{H}$ and $\hat{g}$, respectively, that correspond to the free variables. Let n_{FR} and n_{FX} denote the number of free and fixed variables, respectively. Finally, let C denote the $m \times n_{FR}$ submatrix of $\mathcal{A}$ corresponding to the free variables. The search direction p is chosen so that $x_{FR} + p$ is the solution of the quadratic programming problem with the original objective function restricted to the free variables, subject to the condition that $x_{FR} + p$ exactly satisfies the constraints in the working set. With this definition, p is the solution to the following equality-constraint quadratic programming problem:

$$\text{EQP} \qquad \underset{p}{\text{minimize}} \quad g^T p + \tfrac{1}{2} p^T H p$$

$$\text{subject to} \quad Cp = -r.$$

The optimality conditions for the EQP state the existence of a vector of Lagrange multipliers, denoted by μ, that satisfy

$$C^T \mu = g + Hp. \tag{2.1}$$

The above feasibility and optimality conditions for the EQP can be satisfied by solving the *augmented system* of equations given by

$$\begin{pmatrix} H & C^T \\ C & \end{pmatrix} \begin{pmatrix} p \\ -\mu \end{pmatrix} = K \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ r \end{pmatrix}, \tag{2.2}$$

for p and μ. The major differences among QP methods arise from the numerical methods that solve the augmented system, and the strategies that control changes

in the working set. Some QP methods are equivalent in the sense that they calculate identical sequences of iterates when applied to the same problem. Thus, two algorithms which may appear to be based on different methods, sometimes differ only in the numerical method used to solve the augmented system. (See Djang, 1980, and Best, 1985.) Some of the methods for solving the augmented system are given in Section 2.2. Several strategies that control changes in the working set are discussed in Section 2.3. Following this, initialization procedures are presented in Section 2.4. In Section 2.5, several QP algorithms are summarized—including how they solve the EQP, change the working set, and initialize the problem.

2.2. Solving an EQP

We shall briefly survey several methods for solving the augmented system (2.2). *Lagrangian* or *Kuhn-Tucker* methods use a factorization of the entire matrix K in (2.2) to solve for p and μ. *Projection* methods for solving the augmented system are based upon breaking down the augmented system into several (usually two) simpler systems. The factorizations needed to solve these simpler systems are of a smaller dimension than the factorizations used in the Lagrangian method. Projection methods for solving the augmented system can be further classified as *range-space* or *null-space* methods. This terminology arises because the working set can be viewed as defining two complementary subspaces: the range space of vectors that can be expressed as linear combinations of the rows of C, and the null space of vectors orthogonal to the columns of C. In many cases, the work required to solve the augmented system is directly proportional to the dimension of either the range space or the null space. For example, the methods of Murray (1971), Gill and Murray (1978a), Bunch and Kaufman (1980), and Powell (1981) are null-space methods, and are more efficient when the number of constraints in the working set is close to n, since the dimension of the null space is then relatively small. Range-

space methods increase in efficiency as the number of constraints in the working set decreases. For an example of a range-space method see Gill *et al.*, 1982. The differences in the methods arise because of the different emphasis placed on the relative importance of storage needs, computational efficiency and numerical reliability. When solving large-scale problems, these considerations favor a Lagrangian method, since a projection method may be computationally expensive whenever the number of active constraints is neither close to n nor to zero.

Several examples of projection methods are now given that use an equivalent, but simpler, augmented system. Let S_1 and S_2 be nonsingular $(n_{FR}+m)\times(n_{FR}+m)$ matrices. The solution of (2.2) is equivalent to the solution of

$$S_1 \begin{pmatrix} H & C^T \\ C & \end{pmatrix} S_2 \begin{pmatrix} \bar{p} \\ -\bar{\mu} \end{pmatrix} = -S_1 \begin{pmatrix} g \\ r \end{pmatrix}, \quad \text{with} \quad \begin{pmatrix} p \\ -\mu \end{pmatrix} = S_2 \begin{pmatrix} \bar{p} \\ -\bar{\mu} \end{pmatrix}. \tag{2.3}$$

A particular range-space method follows when S_1 is defined by

$$S_1 = \begin{pmatrix} CH^{-1} & -I \\ I & \end{pmatrix},$$

and S_2 is the identity matrix. The augmented system can then be written as

$$\begin{pmatrix} & CH^{-1}C^T \\ H & C^T \end{pmatrix} \begin{pmatrix} p \\ -\mu \end{pmatrix} = -\begin{pmatrix} CH^{-1}g - r \\ g \end{pmatrix}. \tag{2.4}$$

From (2.4), the following equations for p and μ are obtained:

$$CH^{-1}C^T\mu = CH^{-1}g - r, \tag{2.5a}$$

and

$$Hp = C^T\mu - g. \tag{2.5b}$$

In order to solve (2.5), factorizations of H and $CH^{-1}C^T$ are required.

The remainder of the methods presented in this section will be null-space methods. One choice for S_1 is based on finding an $m \times m$ *reverse* triangular matrix T and an $n_{FR} \times n_{FR}$ matrix Q (see Gill *et al.* 1982) satisfying

$$CQ = (0 \quad T). \tag{2.6}$$

Assume that the columns of Q are partitioned so that

$$Q = (Z \quad Y),$$

where Z is an $n_{FR} \times (n_{FR} - m)$ matrix and Y is an $n_{FR} \times m$ matrix. Define S_2 to be the matrix

$$S_2 = \begin{pmatrix} Z & Y \\ & I \end{pmatrix}, \tag{2.7}$$

and define S_1 as the transpose of S_2. Let p_Z and p_Y denote the first $n_{FR} - m$ and last m elements of $\bar{p}$, respectively. Using (2.7) in (2.3) yields

$$\begin{pmatrix} Z^THZ & Z^THY & \\ Y^THZ & Y^THY & T^T \\ & T & \end{pmatrix} \begin{pmatrix} p_Z \\ p_Y \\ -\mu \end{pmatrix} = - \begin{pmatrix} Z^Tg \\ Y^Tg \\ r \end{pmatrix}. \tag{2.8}$$

Thus, p and μ may be found by successively solving the equations

$$Tp_Y = -r, \tag{2.9a}$$

$$Z^THZp_Z = -Z^Tg - Z^THYp_Y, \tag{2.9b}$$

$$p = Yp_Y + Zp_Z, \tag{2.9c}$$

and

$$T^T\mu = Y^T(g + Hp). \tag{2.9d}$$

In order to solve (2.9), a TQ factorization of C and a factorization of Z^THZ (the *projected Hessian*) are required.

When H is positive definite, another method may be defined using a different choice for the matrix Q in (2.6). Suppose that Q satisfies (2.6), and

$$Q^T H Q = I. \tag{2.10}$$

In this case, we have $Z^T H Z = I$ and $Z^T H Y = 0$. The equations (2.9) for p and μ, therefore, simplify to become

$$T p_Y = -r, \tag{2.11a}$$

$$p_Z = -Z^T g, \tag{2.11b}$$

$$p = Y p_Y + Z p_Z, \tag{2.11c}$$

and

$$T^T \mu = Y^T g + p_Y. \tag{2.11d}$$

Equations (2.11) are used to solve the augmented system in the QP method of Goldfarb and Idnani (1983). Similar techniques have been suggested for both the positive-definite and indefinite cases in which the relationship $Z^T H Z = D$ is maintained instead of (2.10). The matrix D may be diagonal (see Murray, 1971) or block-diagonal (see Bunch and Kaufman, 1980).

If the matrix C has *unit column vectors* (i.e., vectors e_j with the j-th component equal to one, and the remaining components equal to zero), this *special structure* can be used to solve the augmented system more efficiently. An *example* of this special structure is when there are slack variables in the QP. Suppose C has an additional n_s columns that correspond to free slack variables. The $m \times (n_{FR} + n_s)$ matrix C can be written as

$$C = \begin{pmatrix} A_W & \\ A_I & -I \end{pmatrix}, \tag{2.12}$$

where the identity corresponds to the free slack variables. The components of A

corresponding to these slacks are zero, so the augmented system is of the form

$$\begin{pmatrix} H & 0 & A_w^T & A_I^T \\ 0 & 0 & 0 & -I \\ A_w & 0 & 0 & 0 \\ A_I & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} p \\ q \\ -\mu \\ -\lambda \end{pmatrix} = - \begin{pmatrix} g \\ 0 \\ r_w \\ r_I \end{pmatrix}. \tag{2.13}$$

In order to solve (2.13), we can first solve the *smaller* augmented system

$$\begin{pmatrix} H & A_w^T \\ A_w & \end{pmatrix} \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ r_w \end{pmatrix}, \tag{2.14}$$

and then set $\lambda = 0$ and $q = A_I p + r_I$. The smaller augmented system (2.14) can be solved by any one of the methods described earlier.

2.3. Changes to the working set

Two strategies that control changes to the working set are now briefly outlined. Let x denote the current iterate and λ the multipliers of the constraints in the current working set. The strategies described here always attempt to move from the minimum on one set of constraints to the minimum on another by taking steps of the form p and $\mu - \lambda$. The maintenance of certain conditions on the working set causes a shorter step αp and $\alpha(\mu - \lambda)$, where α $(0 \leq \alpha < 1)$ depends upon the active-set strategy being used.

The first strategy discussed is representative of the class of *active-set feasible-point methods*. In this case, x is feasible (i.e., all $r_j \geq 0$) but is not dual feasible (i.e., there exists some constraint for which $\lambda_j < 0$). Changes in the working set are designed to maintain feasibility with respect to the constraints and to move interior to constraints that have negative Lagrange multipliers. At a typical iteration, C comprises the constraints that are satisfied exactly at x (i.e., $r = 0$ in (2.2)). The step length α is equal to one if $x + p$ is feasible (i.e., $r_j + a_j^T p \geq 0$). Otherwise, α is

the largest step such that $r_j + \alpha a_j^T p = 0$ for an index j of a constraint that is not in the working set. A constraint is added to the working set if its residual becomes zero at the step α. A constraint with $\lambda_j < 0$ is selected to be deleted from the working set after a unit step. (The active-set method outlined briefly in Section 2.1 is an active-set feasible-point method.)

The second strategy forms the basis of the class of *dual-feasible active-set methods*. In these methods, x is not feasible (i.e., some $r_j < 0$) but is always dual feasible (i.e., all $\lambda_j \geq 0$). Changes in the working set are designed to maintain non-negative multipliers while moving to satisfy the violated constraints. The step α is equal to one if $x + p$ is dual feasible (i.e., $\mu_j > 0$). Otherwise, α is the largest step such that $\lambda_j + \alpha(\mu_j - \lambda_j) = 0$ for a constraint index j that is in the working set. A constraint is deleted from the working set if its associated value of λ becomes zero at the step α. After a unit step is taken, the iterate will be primal feasible. (If the constraints in the QP were in an equivalent form where all of the general constraints were not necessarily in the working set, then after a unit step is taken, the iterate would satisfy the working set. Then, a constraint with $r_j < 0$ would be selected to be added to the working set.)

For both of these active-set strategies, each change in the working set leads to a simple change to C, which in turn leads to a change in the factorizations being used to solve (2.2). Since the only inequalities in QP1 are bound constraints, only bounds can be added to or deleted from the working set. These changes to the working set lead to changes in the *columns* of C. (When a *bound corresponding to* a *variable* is deleted from or added to the working set, we may shorten the wording and say that the *variable* has been deleted from or added to the working set.) If the only inequalities of a QP were general constraints, only general constraints could be added to or deleted from the working set. These changes to the working set

would lead to changes in the *rows* of the constraint matrix used in the augmented system. If a QP had both bounds and general constraints as inequalities, both bounds and general constraints could be added to or deleted from the working set. These changes to the working set would lead to changes in both the *rows* and *columns* of the constraint matrix used in the augmented system.

2.4. Initialization procedures

Both of the active-set strategies described in the previous section require an *initialization procedure* to obtain an initial primal- or dual-feasible point. It is critical that this procedure be able to utilize a pre-assigned working set. In this way, during later major iterations of an SQP method, it is possible to force the QP subproblems to reach optimality in a *single iteration* because the optimal active set is available before solving the subproblem.

The initialization procedure for the primal feasible-point method is equivalent to a linear programming problem. Consider the sum of infeasibilities $v(p) = \Sigma|r_i|$. Note that $v(p)$ is a linear function of x which is zero at any feasible point, and positive at an infeasible point. Therefore, a feasible point can be found by minimizing $v(p)$.

If a dual-feasible active-set strategy is used, the following initialization procedure may be employed. The procedure is based on finding a subset of the pre-assigned working set on which the multipliers are positive. First, the minimum of the quadratic function on the pre-assigned working set is computed by solving (2.2). If the μ_j are non-negative, the initial point is given by x and $\lambda = \mu$ and the initialization is complete. Otherwise, a constraint with a negative multiplier is deleted, the factorizations are updated and (2.2) is solved again. This process is repeated until (*i*) all the multipliers are positive, in which case x and $\lambda = \mu$ define the required initial point, or (*ii*) the working set is empty, in which case $x + p$ is

the unconstrained minimum and is trivially dual feasible.

An alternative initialization procedure for the dual-feasible active-set method is to always start at the unconstrained minimum (if one exists) and give preference to adding the pre-assigned constraints. However, if the pre-assigned working set is similar to the active set, this scheme is likely to require more work than the procedure above. First, more operations are required to compute the factorizations by updating. Second, even if the pre-assigned working set defines the optimal feasible point, the number of QP iterations may not be equal to the dimension of the optimal working set since it cannot be guaranteed that the multipliers will remain dual-feasible during the intermediate iterations.

2.5. Alternative QP methods

The single-phase method of this thesis is similar to other QP methods. Brief descriptions of several of these methods are now given, in order to facilitate a comparison of them to the single-phase method. Their descriptions will include the method used to solve the EQP, the strategy used to control changes in the working set, and the initialization procedure.

2.5.1. *QPSOL*: a primal-feasible algorithm. *QPSOL* is an active-set feasible-point QP algorithm of Gill, Murray, Saunders, and Wright. In *QPSOL* the Hessian is allowed to be indefinite (see Gill *et al.*, 1983a). In terms of solving the augmented system, *QPSOL* is a null-space projection method that uses equation (2.9). Since it is a feasible-point method, the r and hence p_Y terms in (2.9) are zero. The strategy used by *QPSOL* that controls changes in the working set is described in Section 2.3. *QPSOL* maintains primal feasibility while moving to a constrained stationary point. Then, a constraint with a negative multiplier is deleted, and the algorithm finds a different constrained stationary point with a lower objective value.

The algorithm is also a two-phase method since it first finds a feasible point. In order to perform this initialization procedure, *QPSOL* minimizes the sum of infeasibilities, $v(p)$, that was described in Section 2.4. It minimizes the function $v(p)$ using an active-set strategy that is almost identical to the feasible-point active-set method. The principal differences are that the search direction is defined as $-ZZ^T\nabla v(p)$ and the *largest* value of α is computed, subject to the restriction that the number of violated constraints is not increased. This gives α as the step to the nearest of the satisfied constraints. Several violated constraints may become satisfied during a single iteration. Notice that the feasibility phase does not perform the standard simplex method (i.e., it does not necessarily find a vertex). The implementation of this procedure reflects the similarity of the linear algebraic computations associated with iterations in both the feasibility and QP phases—in particular, each iteration involves an update of the same factorization of the working set. The computations in both phases may be performed by exactly the same program modules. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function. The important feature of this type of implementation is that if the pre-assigned working set is similar to the active set, just a few changes in the working set are necessary to achieve feasibility. In particular, if the initial point is feasbile, the procedure merely computes all the relevant factorizations (which are also needed for the QP phase) and performs a feasibility check.

Complications arise in *indefinite quadratic programming problems*, in which the matrix Z^THZ is indefinite for some Z. In this case, it is not true that any constrained stationary point is a local minimum in the current null space. Furthermore, the direction defined by (2.9*d*) is not necessarily a direction of descent for the quadratic function.

If the iterate is a constrained stationary point with an indefinite projected Hessian, there must exist a vector p_z (a direction of negative curvature) such that $p_z^T Z^T H Z p_z < 0$; hence, if the problem contained no further constraints, the quadratic objective would be unbounded below in the current manifold. However, assuming that the QP has a *finite* solution, a move along such a direction must encounter a constraint that is not already in the working set. Eventually, enough constraints must be added to the working set so that the projected Hessian becomes positive definite.

These observations lead to the manner in which *QPSOL* treats indefiniteness in the projected Hessian. The method finds a positive-definite projected Hessian. When the projected Hessian is positive definite, it may become indefinite only when a constraint is deleted from the working set. If this occurs, *QPSOL* calculates a direction of negative curvature for the search direction and does not delete any further constraints until enough constraints are added to make the projected Hessian positive definite.

2.5.2. Dual-feasible method of Goldfarb and Idnani. The dual-feasible active-set QP method (Goldfarb and Idnani, 1983), referred to in this thesis as *GI*, requires a positive-definite Hessian. In terms of solving the augmented system, *GI* is a range-space projection method that uses equation (2.11). Equation (2.11) is simplified in *GI*, since the r term has exactly one nonzero component and the Zp_z term is always zero. The strategy used by *GI* that controls changes in the working set is described in Section 2.3. *GI* maintains dual feasibility while reducing r to the zero vector. When r becomes the zero vector, the current iterate is optimal and feasible with respect to the current working set. At this point in the algorithm, a violated constraint is added to the working set, making the new r nonzero. The algorithm continues in this way, increasing the value of the primal objective function,

until primal feasibility is achieved. *GI* is a single-phase method, since the origin in the space of dual variables is dual feasible. See Goldfarb and Idnani, 1983, Section 8, for comments on how *GI* can be extended to handle an indefinite Hessian.

GI is a dual method, but it is equivalent to applying an appropriate primal method to the dual QP. It is presented in Goldfarb and Idnani, 1983 and in this thesis as a dual method because, as Goldfarb and Idnani point out, "we believe this to be more instructive". The presentation of *GI* as a dual method will also make clearer the comparison of *GI* to the method presented in this thesis.

2.5.3. Exact-penalty method of Conn and Sinclair. Another method for solving QP's (and other constrained programming problems) is to solve a related, unconstrained problem which has as its objective function the sum of $F(x)$ (the objective function of the QP), and a *penalty* term (see Fletcher, 1981). This term imposes a positive penalty to an iterate which violates any of the constraints of the QP. A commonly-used penalty function is the *absolute value penalty function*, given by

$$P(x, \rho) = F(x) - \rho \sum_i r_i, \quad i \in \mathcal{V}, \tag{2.15}$$

where r_i is the residual of the i-th constraint and $\mathcal{V}$ denotes the set of violated constraints. Let $\bar{x}$ denote a solution of the QP. There is a threshold value $\bar{\rho}$, such that $\bar{x}$ is an unconstrained minimum of (2.15) for any $\rho > \bar{\rho}$. For this reason, penalty functions like $P(x, \rho)$ are sometimes termed *exact* penalty functions. (In contrast, there are differentiable penalty functions for which the solution to the unconstrained problem with the penalty term is not equal to the solution of the constrained problem for any finite ρ. For a discussion of these penalty functions and other exact penalty functions, see Gill *et al.*, 1981 and Fletcher, 1981.) An EQP has a natural step length of unity. However, a step of unity may have no meaning

when using (2.15), because the gradient changes as violated constraints become satisfied. Possible strategies to determine a step length when using (2.15) include a line search to find the minimum along the search direction and determining the first constraint hit along the search direction.

The QP method by Conn and Sinclair, 1975 (referred to in this thesis as *CS*) uses the exact penalty function (2.15). The search direction and multipliers are determined by solving (2.9). Even though *CS* solves an unconstrained problem, there is an underlying working set which determines the penalty term and the null space, represented by the matrix Z. The strategy to control changes to the working set is similar to the active-set feasible-point strategy. It is similar since the constraint with the most negative multiplier is deleted from the working set. However, the line search (which determines the minimum along the search direction) determines which constraints to add to the working set. If the new iterate satisfies an additional constraint, it is added to the working set. Note that the new iterate may violate one or more previously satisfied constraints. Since *CS* is not a primal- or dual-feasible method, an initialization phase to find a feasible point is not required. Instead, *CS* must initialize the value of ρ, and also determine an upper bound on ρ, at which the QP problem is considered infeasible, if the solution of (2.15) has a positive penalty.

Chapter Three

The Single-Phase Method

3.1. Outline of the single-phase method

The single-phase method of this thesis is an active-set method. In order to solve the augmented system, the single-phase method can use the Lagrangian method, or, if the problem is not large, a projection method. In this chapter, the single-phase method uses the Lagrangian method so that it is applicable to large-scale problems.

The form of the quadratic programming problem solved is QP1. Since the only inequalities in QP1 are bound constraints, only bounds enter or leave the working set. These changes to the working set lead to column changes in C.

The main steps of the single-phase method are:

1. **Initialise.** Start with any $x \geq 0$.
2. **Test for convergence.**
3. **Find p and μ.** Using a Lagrangian method, solve

$$K \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ r \end{pmatrix}, \qquad \text{where} \qquad K = \begin{pmatrix} H & C^T \\ C & \end{pmatrix}.$$

4. **Either delete a bound or calculate the step length.**
   a. If a bound is deleted, update K and return to step 2.
   b. If the step length is calculated, continue with step 5.
5. **Update.** $x \leftarrow x + \alpha p$, $r \leftarrow (1-\alpha)r$, and $g \leftarrow (1-\alpha)g + \alpha C^T \mu$.
   If $\alpha < 1$, add a bound and update K.
   Return to step 2.

The iterates of this single-phase method may be infeasible in the sense that the general constraints may be violated. However, the bound constraints will *always* be satisfied. Thus, the bound constraints in the working set are satisfied, but the general constraints in the working set may be satisfied or *violated*. However, the violated constraints are satisfied at $x + p$ (i.e., at a unit step). (Note that if the current iterate is not feasible, the search direction is *not necessarily* a *descent direction*.) With each nonzero step taken, the magnitude of the residual of each violated constraint decreases, so that eventually a feasible point is found (when one exists). Once a feasible point is found, the method is equivalent to an active-set feasible-point method.

In order to solve the augmented system, the matrix K must be nonsingular. In an active-set feasible-point method (see Section 2.3), K can become singular only if the projected Hessian becomes singular. However, in the single-phase method of this thesis, K can become singular if either the projected Hessian becomes singular *or* if the constraints become dependent. A general strategy that maintains nonsingularity in K as constraints are added and deleted is presented in Section 3.2. The single-phase Lagrangian method will be described in more detail in Section 3.3.

3.2. Background: singularity in the augmented system

If a change in the working set results in a singular K, a strategy must be used that will restore nonsingularity. If a bound is hit (added to the working set) the dimension of the projected Hessian decreases by one. If the projected Hessian is positive definite, the new projected Hessian is necessarily positive definite. If the projected Hessian is not positive definite, the new projected Hessian may be singular. Thus, if a bound is hit and the new K is singular, either the projected Hessian is singular *or* the constraints are dependent. In this event the Singularity Rule (SR) will be used to determine a K that is nonsingular. (The Singularity Rule will be described in the next subsection.) If a variable is released from its bound (deleted from the working set) and the new K is singular, the projected Hessian is singular. If this occurs, a strategy will be employed that makes a slight perturbation to the Hessian to determine a K that is nonsingular. These two strategies are described in the next two subsections.

3.2.1. Adding a bound. When a variable is added to the working set, thereby fixing the variable on its bound, either the projected Hessian is singular or the constraints are linearly dependent.

The search direction is calculated by solving the augmented system of equations

$$K \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ r \end{pmatrix}, \tag{3.1}$$

where K is nonsingular. Suppose that the step α along the search direction causes the k-th component of x_{FR} to hit its bound. When the k-th variable is added to the working set, the matrix of the new augmented system can be represented by

$$M = \begin{pmatrix} H & C^T & e_k \\ C & & \\ e_k^T & & \end{pmatrix}. \tag{3.2}$$

The method cannot continue if M is singular. Note, however, that if M is singular, it can have only *one* zero eigenvalue. Consequently, it is possible to maintain nonsingularity in the matrix of the augmented system by freeing one of the currently fixed variables (other than the one which was just fixed).

Therefore, when a bound is hit, it must be determined whether or not M is singular. One method to identify singularity in M is to form its factorization. Another method is to use the following Lemma.

Lemma 1. *Let y and z solve*

$$\begin{pmatrix} H & C^T \\ C & \end{pmatrix} \begin{pmatrix} z \\ y \end{pmatrix} = \begin{pmatrix} e_k \\ 0 \end{pmatrix}. \tag{3.3}$$

The matrix M is singular if and only if $e_k^T z = 0$.

Proof:

$\Rightarrow$) If M is singular, there exist vectors $\bar{y}$ and $\bar{z}$ that satisfy

$$\begin{pmatrix} H & C^T \\ C & \\ e_k^T & \end{pmatrix} \begin{pmatrix} \bar{z} \\ \bar{y} \end{pmatrix} = \begin{pmatrix} -e_k \\ 0 \\ 0 \end{pmatrix}. \tag{3.4}$$

However, z and y are the unique solutions to (3.3), so $\bar{z} = -z$, $\bar{y} = -y$, and $e_k^T z = 0$.

$\Leftarrow$) If $e_k^T z$ is zero, z and y solve (3.4). Existence of a solution to (3.4) implies that M is singular. ∎

A different proof for Lemma 1 explicitly uses the inertia of M and K. The inertia of a real symmetric matrix M is the triple

$$\text{In}(M) = (\pi, \nu, \delta),$$

where π, ν, and δ denote respectively, the number of positive, negative, and zero eigenvalues of M. A well known expression for the inertia is

$$\text{In}(M) = \text{In}(K) + \text{In}(M/K), \tag{3.5}$$

where (M/K) is the Schur complement of K in M (see Haynsworth, 1968). Using this formula, if K is nonsingular, M is singular if and only if (M/K) equals zero. Using the definition of the Schur complement (see Cottle, 1974) gives

$$(M/K) = -(e_k^T \quad 0)K^{-1}(e_k^T \quad 0)^T = -e_k^T z. \tag{3.6}$$

Thus, the singularity of M may be ascertained by solving (3.3), and calculating $e_k^T z$. If M is singular, a variable must be freed that ensures (*i*) the matrix in the new augmented system is nonsingular, and (*ii*) the new search direction is a feasible direction with respect to the new free variable. (The vector p is a *feasible direction* with respect to constraint j, if any positive step along p moves to a point which satisfies constraint j.) The SR will identify a variable to free, such that these two conditions will be met. Let the vector a denote the column of A corresponding to the new free variable. Let the vector h denote the row and column of $\mathcal{H}$, corresponding to the new free variable. Let the constant h_{jj} denote the diagonal element of $\mathcal{H}$ corresponding to the new free variable. The new H is given by

$$\begin{pmatrix} H & h \\ h^T & h_{jj} \end{pmatrix}.$$

At a step of α along p, the new residual and new gradient, denoted by $\bar{r}$ and $\bar{g}$ respectively, can be expressed as

$$\bar{r} = (1-\alpha)r, \tag{3.7a}$$

and

$$\bar{g} = (1-\alpha)g + \alpha C^T \mu. \tag{3.7b}$$

The new augmented system, formed by adding the k-th variable to the working set and deleting the variable corresponding to column a from the working set, can

then be written as

$$\begin{pmatrix} H & C^T & h & e_k \\ C & & a & \\ h^T & a^T & h_{jj} & \\ e_k^T & & & \end{pmatrix} \begin{pmatrix} \bar{p} \\ -\bar{\mu} \\ \beta \\ -\theta \end{pmatrix} = - \begin{pmatrix} \bar{g} \\ \bar{r} \\ \gamma \\ 0 \end{pmatrix}. \tag{3.8}$$

The new search direction is $(\bar{p}^T, \beta)^T$. The new free variable is at its lower bound of zero, so if β is nonnegative, the new search direction is a feasible direction with respect to the new free variable. Thus, the SR has found a suitable variable to free if (*i*) the matrix in (3.8) is nonsingular, and (*ii*) β is nonnegative.

A simple means to determine β when the matrix in (3.8) is nonsingular will now be given. The explanation is simplified by constructing an equivalent, but simpler system, using the transformation expressed by equation (2.3). Let S_1 be the identity matrix except that the bottom row is given by

$$-(z^T \quad y^T \quad 0 \quad -1), \tag{3.9}$$

where z and y solve (3.3). Let S_2 be the transpose of S_1. Define the constant k such that

$$k = a^T y + h^T z. \tag{3.10}$$

By using Lemma 1, the transformed augmented system, constructed from (3.8), can be written as

$$\begin{pmatrix} H & C^T & h & \\ C & & a & \\ h^T & a^T & h_{jj} & -k \\ & & -k & \end{pmatrix} \begin{pmatrix} \hat{p} \\ -\hat{\mu} \\ \beta \\ -\theta \end{pmatrix} = -S_1 \begin{pmatrix} \bar{g} \\ \bar{r} \\ \gamma \\ 0 \end{pmatrix}, \tag{3.11}$$

$$\text{where} \qquad S_2 \begin{pmatrix} \hat{p} \\ -\hat{\mu} \\ \beta \\ -\theta \end{pmatrix} = \begin{pmatrix} \bar{p} \\ -\bar{\mu} \\ \beta \\ -\theta \end{pmatrix}. \tag{3.12}$$

The matrix in (3.11) is nonsingular if and only if k *is not equal to zero.* (By expanding along the last column and then the last row, the determinant is equal to $\pm k^2 \det(K)$.) If k is not equal to zero, (3.11) can be solved for β giving

$$-k\beta = z^T\bar{g} + y^T\bar{r}. \tag{3.13}$$

Rewriting (3.13) gives

$$\beta = -\frac{y^T\bar{r} + z^T\bar{g}}{a^Ty + h^Tz}. \tag{3.14}$$

Thus, the SR consists of searching for a variable to free by calculating k and β for each fixed variable. If k is nonzero and β is nonnegative for any fixed variable, the variable is a suitable variable to free.

3.2.2. Deleting a bound. Consider singularity in K when a variable is deleted from the working set, thereby freeing the variable from its bound. In this case, the projected Hessian must be singular.

The search direction is calculated by solving (3.1). Suppose that the k-th variable is deleted from the working set. The matrix of the new augmented system is then given by

$$M = \begin{pmatrix} H & C^T & h \\ C & & a \\ h^T & a^T & h_{kk} \end{pmatrix}, \tag{3.15}$$

where a denotes the k-th column of A, h denotes the k-th row and column of H, excluding the k-th element, and h_{kk} denotes the k-th diagonal element of H. The method cannot continue when M is singular. Note, however, that if M is singular, it can have only *one* zero eigenvalue. Consequently, it is possible to maintain nonsingularity in the matrix of the augmented system by modifying h_{kk} by a sufficiently small positive amount, ϵ.

Therefore, when a bound is hit, it must be determined whether or not M is singular. One method to identify singularity in M is to form its factorization. Another method is to solve for the Schur complement of K in M, (M/K). In this case,

$$(M/K) = h_{kk} - \begin{pmatrix} h^T & a^T \end{pmatrix} K^{-1} \begin{pmatrix} h^T & a^T \end{pmatrix}^T. \tag{3.16}$$

M is singular if and only if (M/K) equals zero. Thus, the singularity of M may be ascertained by calculating (M/K).

Note that if the search direction is nonzero, a variable *need not be deleted.* This implies that if a variable is deleted when the search direction is nonzero and the new augmented system is singular, we can avoid singularity difficulties by *not deleting the variable.* If the search direction is the zero vector, the current iterate is a feasible, constrained stationary point. In this event, a variable must be deleted in order to continue. If singularity results, a strategy must be employed that will restore nonsingularity, and determine a descent direction. The following theorem shows how this can be accomplished.

Theorem 1. *Assume that the current iterate is a feasible, constrained stationary point. Let the k-th variable be on its bound and λ_k (< 0) denote the corresponding multiplier. Suppose that when the k-th bound is deleted from the working set, M (3.15) is singular. Form $\bar{M}$ by adding a small positive quantity ϵ to h_{kk} (i.e., $\bar{h}_{kk} = h_{kk} + \epsilon$). If the augmented system is solved using $\bar{M}$, the search direction is a descent direction.*

Proof:

The search direction p, is the zero vector, since the current iterate is a feasible, constrained stationary point. The multiplier, λ_k, satisfies the equation

$$\lambda_k = a^T \mu + g_k, \tag{3.17}$$

where g_k is the k-th component of the gradient, a is the k-th column of A, and (p, μ) solves (3.1), with $p = 0$.

Since M is singular, (M/K) equals zero. By modifying h_{kk}, we force $(\bar{M}/K)$ to have the value ϵ. This implies that $\bar{M}$ is nonsingular. Using $\bar{M}$, the augmented system is given by

$$\begin{pmatrix} H & C^T & h \\ C & & a \\ h^T & a^T & \bar{h}_{kk} \end{pmatrix} \begin{pmatrix} \hat{p} \\ -\bar{\mu} \\ p_k \end{pmatrix} = \begin{pmatrix} -g \\ 0 \\ -g_k \end{pmatrix}, \tag{3.18}$$

where p_k and g_k are, respectively, the k-th components of the new search direction, $\bar{p}$, and the new gradient, $\bar{g}$. If p_k equals zero, then $\bar{\mu}$ equals μ, and $\hat{p}$ is the zero vector, since they must satisfy the original augmented system. This would imply that

$$a^T\mu = g_k. \tag{3.19}$$

Equations (3.17) and (3.19) together imply that λ_k equals zero, which contradicts the hypothesis. Thus, p_k must be nonzero.

The projected gradient can be expressed by

$$\bar{p}^T\bar{g} = p_k\lambda_k. \tag{3.20}$$

By solving (3.18) and then reversing the sign of $\bar{p}$, if p_k is negative, equation (3.20) implies that $\bar{p}^T\bar{g}$ is negative. ∎

A small value for ϵ implies that the matrix in (3.18) is ill conditioned. The search direction can be calculated independent of ϵ (see Section 3.3.4).

A different scheme could be employed when deleting a bound results in a singular M. The scheme is to delete a *different* bound—one that does not result in a singular M. However, it is certainly possible that a singular M would result if any one of the bounds in the working set were deleted. For this reason, this scheme is not to be recommended.

In summary, if the current iterate is a feasible, constrained stationary point and the augmented matrix is singular after a variable with a negative multiplier is deleted, the strategy of making a slight perturbation to H will be used. This strategy is appropriate for use when the augmented system is solved using the Lagrangian method. A similar, but slightly different, perturbation to H will be used in the implementation of the single-phase method described in the next chapter.

3.3. Single-phase Lagrangian method

Important details of how the single-phase method may be implemented using the Lagrangian method to solve the EQP will now be described. First, the problem of singularity in K will be resolved. Second, a means of detecting that the problem has no feasible point will be explained. Next, convergence of the method to an optimal point will be proven. An efficient means of solving successive large-scale augmented systems follows.

3.3.1. Resolution of singularity. Singularity in K can occur when a variable is either deleted from, or added to, the working set. In order to detect singularity, either of the two alternative strategies outlined in the previous section may be used. Following is a brief description of a computational feature of each of these strategies. The first strategy calculates a factorization of M. *If* M is singular, either the SR is used to search for a suitable variable to free from its bound, or a slight perturbation to H is made. Note that if M is singular when a variable is added to the working set, a factorization of K must be *restored*. Thus, the work to find a factorization of M and then restore a factorization of K has only provided the information that M is singular. The second strategy consists of finding (M/K). If (M/K) is not zero, M is nonsingular. In this event, M is known to be nonsingular before its factorization is formed, but the work to find (M/K) has only provided the information that M is nonsingular. If (M/K) is zero, M is singular, and either

the SR is used to search for a suitable variable to free or a slight perturbation to H is made. Both of these strategies of maintaining nonsingularity in K require a significant amount of computation just to determine if M is singular.

If singularity occurs when a variable is deleted from the working set, the strategy of modifying the Hessian discussed in the previous section, may be used in a straightforward manner.

If the addition of the k-th variable to the working set causes singularity, the SR will be used. If the SR finds a variable to free from its bound, a search direction can be computed and the method can continue. If the SR cannot find a suitable variable, the problem *might* not have a feasible point. In order to determine if the problem does not have a feasible point, the k-th variable is not added to the working set. Instead, artificial bounds are added one at a time, keeping C independent. (The artificial bound corresponds to the current value of the variable.) After each artificial bound is added, an attempt is made to add the k-th variable to the working set. If M is singular, the SR is used again. If it does not find a suitable variable to free, another artificial bound is added. As these artificial bounds are added, either (i) the k-th variable will be successfully added to the working set, or (ii) the projected Hessian will become positive definite. Eventually, the projected Hessian must become positive definite. If the projected Hessian is positive definite, M is singular when the k-th variable is added, and the SR still fails to find a suitable variable to delete, the problem does not have a feasible point (see Section 3.3.2).

3.3.2. Detection of an infeasible problem. The inability of the SR to find a suitable variable to delete does not necessarily imply that there is no feasible point. However, if the projected Hessian is positive definite and hitting a bound results in a singular M, then it will be shown that the new C must be rank deficient. In this case, if the SR does not find a suitable constraint, the problem is infeasible. Before

proving this, some preliminary Lemmas are required.

In Lemmas 2, 3, 4, and 5, and Theorem 2, the following assumptions are made: (i) the matrices H and C from (3.1) have been used to calculate Z^THZ, where Z satisfies $CZ = 0$, and (ii) z and y solve (3.3).

Lemma 2. *If M (3.2) is singular and Z^THZ is positive definite, then z is zero.*

Proof: Write z in terms of its range-space and null-space components as

$$z = Z\varsigma_z + Y\varsigma_Y.$$

Equation (3.3) gives

$$0 = Cz = CZ\varsigma_z + CY\varsigma_Y = CY\varsigma_Y.$$

CY is nonsingular so ς_Y is zero, and z can be written as $z = Z\varsigma_z$. Using (3.3) again gives

$$C^Ty + Hz = e_k. \tag{3.21}$$

Premultiply (3.21) by z^T giving

$$\varsigma_z^TZ^TC^T + \varsigma_z^TZ^THZ\varsigma_z = z^Te_k. \tag{3.22}$$

CZ is zero, and Lemma 1 implies that z^Te_k is zero, so (3.22) simplifies to

$$\varsigma_z^T(Z^THZ)\varsigma_z = 0. \tag{3.23}$$

Since Z^THZ is positive definite, ς_z is zero, which implies that z is zero. ∎

Lemma 3. *If M (3.2) is singular and Z^THZ is positive definite, then $e_k^TZ = 0$.*

Proof: Lemma 2 and (3.3) imply that

$$C^Ty = e_k. \tag{3.24}$$

Premultiplying (3.24) by Z^T gives

$$Z^TC^Ty = Z^Te_k. \tag{3.25}$$

Since CZ equals zero, (3.25) implies that e_k^TZ is zero. ∎

Lemma 4. *If $e_k^T Z = 0$, then M (3.2) is singular.*

Equation (3.3) implies that Cz is zero. This implies that z can be written as $z = Z\zeta_z$. Premultiplying this by e_k^T implies $e_k^T z$ is zero, since $e_k^T Z$ is zero. By Lemma 1, M is singular, since $e_k^T z$ equals zero. ∎

As the next lemma demonstrates, singularity in M can be attributed to either the projected Hessian or the constraints. Let C_k denote the matrix C with the k-th column, c_k, deleted.

Lemma 5. *C_k is rank deficient if and only if $z = 0$.*

Proof:

$\Rightarrow$) Rank deficiency of C_k implies the existence of a nonzero vector, $\bar{y}$, such that $C_k^T \bar{y}$ is zero. The matrix C having full rank implies that $c_k^T \bar{y}$ is nonzero. Together, these facts imply the existence of a vector y which satisfies $C^T y = e_k$. This implies that z is zero.

$\Leftarrow$) z being the zero vector implies the existence of a vector y which satisfies $C^T y = e_k$. The vector y, therefore, also satisfies $C_k^T y = 0$. This implies that C_k is rank deficient. ∎

Theorem 2. *Suppose that $Z^T H Z$ is positive definite, the search direction hits the k-th bound with an α less than one, and M (3.2) is singular. If the SR calculates that either k is zero or β is negative for each fixed variable, then there is no feasible point.*

Proof: Farkas' Theorem of the Alternative states that exactly one of the two following alternatives is true:

(*i*) there exists a vector x satisfying $Ax = b$ and $x \geq 0$;

(*ii*) there exists a vector π satisfying $\pi^T A \leq 0$ and $\pi^T b > 0$.

The proof will demonstrate that the SR has constructed a π satisfying (*ii*).

Lemma 2 implies that z is the zero vector. This implies that y solves (3.24). Premultiplying (3.24) by p^T gives

$$p^T C^T y = p^T e_k. \tag{3.26}$$

By using $Cp = -r$ and $\bar{r} = (1-\alpha)r$ (equations (3.1) and (3.7a), respectively), equation (3.26) can be simplified to

$$y^T \bar{r} = (\alpha - 1)p_k, \tag{3.27}$$

where p_k is the k-th component of p. The k-th bound was hit by p, so p_k is negative. Thus, equation (3.27) implies that $y^T\bar{r}$ is positive.

Since z is the zero vector, the definitions of k and β given in (3.10) and (3.14), respectively, simplify to $k = a^T y$, and

$$\beta = -\frac{y^T \bar{r}}{y^T a}. \tag{3.28}$$

If k is zero, β is undefined. By hypothesis, β is negative or undefined for all fixed variables. This fact, together with the definition of β given in (3.28), and $y^T\bar{r}$ being positive, implies that $y^T a$ is nonnegative for *all* possible vectors a.

Recall that A can be partitioned into free and fixed components by

$$A = \begin{pmatrix} A_{FR} & A_{FX} \end{pmatrix} = \begin{pmatrix} C & A_{FX} \end{pmatrix}. \tag{3.29}$$

Let $\pi = -y$. Using (3.29) and (3.24) gives

$$\pi^T A = -y^T A = -y^T \begin{pmatrix} C & A_{FX} \end{pmatrix} = -\begin{pmatrix} e_k^T & y^T A_{FX} \end{pmatrix}. \tag{3.30}$$

It has been shown that $y^T a$ is nonnegative for every column of A_{FX}. Together with (3.30), this implies that

$$\pi^T A \leq 0. \tag{3.31}$$

Let $\bar{x}_{FR} = x_{FR} + \alpha p$. Using the definition of r, and equations (3.24) and (3.29) gives

$$\begin{aligned} \pi^T b &= -y^T(A\bar{x} - \bar{r}) \\ &= -y^T C \bar{x}_{FR} - y^T A_{FX} \bar{x}_{FX} + y^T \bar{r} \\ &= -e_k^T \bar{x}_{FR} - y^T A_{FX} \bar{x}_{FX} + y^T \bar{r}. \end{aligned} \tag{3.32}$$

The k-th component of x_{FR} equals zero, $\bar{x}_{FX}$ equals zero, and $y^T\bar{r}$ is positive, so (3.32) implies that

$$\pi^T b > 0. \tag{3.33}$$

Equations (3.31) and (3.33) satisfy alternative (*ii*), so there is no feasible point. ∎

3.3.3. Finite convergence to an optimal point. Since the working set contains all the general constraints, a feasible point has been found when r equals zero. During every iteration of the method, r is reduced to $(1 - \alpha)r$. If α is positive for every iteration, r converges to zero and a feasible point will have been found.

Before proving that the algorithm will find a feasible point, if one exists, two assumptions must be made.

1. The initial point has at least m positive components, and the initial working set includes all zero-valued variables.
2. Movement along the search direction hits at most one bound at the step of α.

The first assumption can easily be met by making at least m components of the initial point positive, and including all zero-valued variables in the initial working set. The second assumption (the *nondegeneracy assumption*), includes the more standard nondegeneracy assumption that movement along the search direction will not hit a point that exactly satisfies linearly dependent constraints. However, the nondegeneracy assumption used here is only slightly more restrictive than the standard one. The standard assumption can be equivalently stated as: *If the search*

direction has d degrees of freedom, movement along it can hit at most d constraints simultaneously (such that the new iterate does not satisfy linearly dependent constraints). The assumption used here can be equivalently stated as: *If the search direction has d degrees of freedom, movement along it can hit at most one bound at the step of* α. Thus, when d is one, the assumptions are equivalent. (Since the initial point is arbitrary, the second assumption could be enforced simply by perturbing the variables corresponding to all but one of the bounds hit.)

A pseudo-Fortran version of how the single-phase method finds a feasible point is given in Table 1. This version, which is used in the following proof, permits variables to be deleted from the working set only by a constraint exchange performed as the result of using the SR.

Theorem 3. *The algorithm will find a feasible point, or indicate that no feasible point exists, in a finite number of steps.*

Proof: By the first assumption, the initial α must be nonzero, since all zero-valued variables are in the working set. If α is one, the next iterate is feasible. If α is less than one, by the second assumption, only one additional variable has a zero value at the next iterate. After each pass through the inner repeat loop, one of three things must happen:

1. The loop is terminated because the SR finds a suitable constraint to delete. In this case, a constraint exchange occurs. Since the SR was used, the next search direction calculated is a feasible direction with respect to the freed variable. All zero-valued variables (except the one freed) are in the new working set so the next α is positive. This implies that the value of the freed variable will be positive at the next iterate.

2. The loop is terminated because the SR did not find a suitable variable to delete and the projected Hessian is positive definite. In this case, Theorem 2 implies that no feasible point exists.

3. The loop is repeated because the SR did not find a suitable variable to delete and the projected Hessian is not positive definite. The bound hit is not added to the working set. In this case an artificial bound is added (such that C remains linearly independent).

Note that when the SR calculates β for these artificial bounds in later iterations, β can be positive or negative, since any direction is feasible with respect to an artificial bound. After a finite number of artificial bounds are added, the projected Hessian must become positive definite. Thus, after a finite number of passes through the inner repeat loop, case 1 or case 2 must occur.

Because α is always positive, the residuals will converge to zero and a feasible point will be found, if one exists. This convergence to a feasible point will occur in a finite number of steps, if the feasible region is bounded. An infinite sequence in a closed, bounded region has at least one limit point. Let $\bar{x}$ denote a limit point of the algorithm. There exists a positive radius determining a sphere about $\bar{x}$ such that the only constraints within the sphere pass through $\bar{x}$. Denote this sphere by S. After a finite number of steps, an iterate will enter S. Once an iterate lies in S, all the bounds in the working set are satisfied exactly at the current iterate and at $\bar{x}$. The residuals *decrease* monotonically from their value at the current iterate to zero at $\bar{x}$. Movement to a point outside S from a point in S would *increase* the residuals. This implies that once the iterates enter S, they will remain in S. The bounds hit inside S are bounds that are satisfied exactly at $\bar{x}$. Therefore, after a finite number of bounds are hit inside S, the iterates will converge to $\bar{x}$. Thus, the

Table 1

Pseudo-Fortran version of finding a feasible point

```
repeat
        compute p, α
        if ( bound is hit ) then
              repeat
                    if ( new K is singular ) then
                          if ( SR finds constraint to delete ) then
                                exchange constraints
                          else
                                if ( projected Hessian is positive definite ) then
                                      error = true
                                else
                                      add artificial bound
                                end if
                          end if
                    else
                          add constraint
                    end if
              until ( new K is nonsingular or error )
        end if
        x_FR = x_FR + αp
        feas = ( iterate satisfies all constraints )
until ( feas or error )
```

algorithm will find a feasible point, if one exists, in a finite number of steps. ∎

Once a feasible point has been found, the single-phase method becomes an active-set feasible-point method which will determine an optimal point. The proof of this depends upon the strategy used to control changes to the working set. For example, when the implementation of the single-phase method described in the next chapter finds a feasible point, the implementation and *QPSOL* become essentially the same algorithm. For a convergence proof, see Section 4.8.

3.3.4. Solving the augmented system. In order to be applicable to large-scale problems, the single-phase method uses a Lagrangian method to solve the augmented system. One of the fastest methods for solving a *single* system of the

form (3.1) is by use of sparse matrix factorization techniques (see e.g., Duff and Reid, 1984; George and Liu, 1980). However, within the context of a single QP iteration, where just a single column of the matrix C may change, the factorization of a sparse system from scratch would clearly be inefficient. We shall now describe how a *single* sparse factorization of a system of the form (3.1), when used in conjunction with the updated factorization of a small dense matrix, may be used to compute p and μ for many (possibly all) QP iterations.

Equation (3.1) will be solved using the *Schur complement update* (see, e.g., Bisschop and Meeraus, 1977, 1980, Gill *et al.*, 1984a). Given factorizations of the matrices B and $S = N - U^T B^{-1} U$ (the *Schur complement*), the solution of the partitioned system

$$\begin{pmatrix} B & U \\ U^T & N \end{pmatrix} \begin{pmatrix} v \\ \eta \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix} \tag{3.34}$$

can be determined by solving, in turn, the equations:

$$Bw = b; \tag{3.35a}$$

$$S\eta = d - U^T w; \tag{3.35b}$$

$$Bv = b - U\eta. \tag{3.35c}$$

We now consider how to define the quantities in (3.34) corresponding to (3.1). The symmetric matrix B is the partitioned $(n_{FR} + m)$-dimensional matrix, K. The vector μ is the last m components of v, and b is defined by

$$b = -\begin{pmatrix} g \\ r \end{pmatrix}.$$

The vector p is composed of selected elements of η and v. Each change in the working set leads to a new column in U (row in U^T). The definition of the column of U and the corresponding element of d depends on whether a bound is added or deleted.

We illustrate the definition of U, N and d using the first iteration as an example. Suppose that a bound indexed by j, with the vector a denoting the corresponding column of A, is deleted from the working set so that

$$\bar{C} = (C \quad a).$$

Let the vector h denote the j-th row and j-th column of H, excluding the j-th element, and the constant h_{jj} denote the j-th diagonal element of H. Let g_j denote the j-th component of the gradient vector.

The vector v_1 and η_1 then satisfy

$$\begin{pmatrix} \boxed{\begin{matrix} H & C^T \\ C & 0 \end{matrix}} & \begin{matrix} h \\ a \end{matrix} \\ \begin{matrix} h^T & a^T \end{matrix} & h_{jj} \end{pmatrix} \begin{pmatrix} v_1 \\ -\eta_1 \end{pmatrix} = - \begin{pmatrix} g \\ r \\ g_j \end{pmatrix}, \tag{3.36}$$

and may be found from (3.35) with

$$N = h_{jj} \qquad \text{and} \qquad U = \begin{pmatrix} h \\ a \end{pmatrix}.$$

Suppose, on the other hand, that the first free variable in C is added to the working set. The system to be solved is then

$$\begin{pmatrix} \boxed{\begin{matrix} H & C^T \\ C & 0 \end{matrix}} & \begin{matrix} e_1 \\ 0 \end{matrix} \\ \begin{matrix} e_1^T & 0 \end{matrix} & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ -\eta_1 \end{pmatrix} = - \begin{pmatrix} g \\ r \\ 0 \end{pmatrix}, \tag{3.37}$$

where e_1 is the first coordinate vector. Note that the effect of the vector e_1 and the zero component in the right-hand side of (3.37) is to force the first component of v_1 to be zero. The vector v_1 and η_1 are computed from (3.35) with

$$N = 0 \qquad \text{and} \qquad U = \begin{pmatrix} e_1 \\ 0 \end{pmatrix}.$$

With this approach, the procedure for computing p and μ after each change in the working set is the same whether a bound is added or deleted. In either case, the change consists of appending another row to U, row and column to N, and element to d.

Theorem 1 modifies h_{jj} by adding ϵ when the matrix in (3.18) is singular. We shall now show how the search direction can be calculated independent of ϵ. In order to conform to the nomenclature of this section, equation (3.36) will be used instead of (3.18).

When the matrix is modified by adding ϵ to h_{jj}, S equals ϵ. Thus, equation (3.35b) gives

$$\epsilon\eta = -g_k - \begin{pmatrix} h \\ a \end{pmatrix}^T \begin{pmatrix} p \\ \mu \end{pmatrix} = c_1.$$

Equation (3.35c) gives

$$Bv = \begin{pmatrix} g \\ r \end{pmatrix} - \begin{pmatrix} h \\ a \end{pmatrix} \eta.$$

Let

$$\nu = \begin{pmatrix} v \\ \eta \end{pmatrix}.$$

Since ϵ can be chosen arbitrarily small, the limit of $\nu/\|\nu\|$ as $\epsilon \to 0$ can be used as the solution of (3.36). This limit is given by

$$\begin{pmatrix} q \\ c_1 \end{pmatrix},$$

where

$$Bq = \begin{pmatrix} h \\ a \end{pmatrix}.$$

The essential operations in (3.35) are two solves with B and a solve with S. If the number of iterations is small enough (say, less than 100), S may be treated as a

dense matrix. It is then straightforward to use an orthogonal factorization $QS = R$ ($Q^TQ = I$, R upper triangular) or an analogous factorization $LS = U$ based on Gaussian elimination (L square, U upper triangular). These factorizations can be maintained in a stable manner as S is updated to reflect changes to C. (The updates involve adding rows and columns of S; see Gill and Murray, 1974).

An implementation of this method requires the availability of a symmetric indefinite sparse equation solver (e.g., the subroutine MA27, Duff and Reid, 1984) and a procedure for updating the LU or LQ factors of the Schur complement as rows and columns are added. We have described how to add bounds to the working set that are not initially in the working set. If we wish to add a bound that was initially in the working set, i.e., add a row to U, this addition may be achieved instead by deleting the row from U that was added in order to delete the bound from the initial working set. This will have the beneficial effect of reducing the dimension of the Schur complement. Similarly, if we wish to delete a bound that was not initially in the working set, i.e., add a row to U, this deletion may be achieved instead by deleting the row from U that was added in order to add the bound to the initial working set. (It may be worthwhile to favor such changes to the working set, since they decrease the dimension of S.)

As iterations proceed, the dimension of the Schur complement will grow and the work required in (3.35*b*) increases with every change in the working set. As a consequence, it will be periodically necessary to abandon the current Schur complement and factorization of B. At this point the matrix B may be redefined with constraints from the latest working set and refactorized. (The situation is similar to that in linear programming, where product forms of the basis factors must be periodically condensed.) The number of updates that need be performed before a refactorization becomes worthwhile will vary with each problem. While the optimal

frequency number may vary considerably from problem to problem we anticipate that typically 100 updates will be possible before refactorization.

The normal procedure for a QP algorithm is to solve for the search direction p directly. We shall show that the Schur complement is more efficiently implemented by solving for q, where $x_0 + q$ is the minimum on the working set defined by C. The improved efficiency is derived from the observation that the right-hand side vector b will be *constant* for all iterations. Consequently, the vector w need be computed only once, and only the last component of $U^T w$ changes at each iteration. This reduces the work required to solve (3.34) by approximately half.

Instead of solving (3.1), the system

$$\begin{pmatrix} H & C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} q \\ -\mu \end{pmatrix} = - \begin{pmatrix} g(x_0) \\ r(x_0) \end{pmatrix}, \tag{3.38}$$

is solved instead, where x_0 is the initial point. The vector p can be determined from the relationship

$$p = x_0 - x + q.$$

When a variable is deleted from the working set, b (3.34) stays constant and a single component is appended to d (see (3.36)). However, if (3.37) is used when a variable is added to the working set, b does not remain constant. In order to allow b to remain constant in this case, the right-hand side of (3.37) can be replaced by

$$- \begin{pmatrix} g \\ r \\ r_1 \end{pmatrix},$$

where r_1 is the residual (at x_0) of the variable added to the working set. The effect of the vector e_1 and the component r_1 is to force $x_0 + q$ to satisfy exactly the bound added to the working set. Thus, the dimension of d increases, but b remains *constant* when we solve for q instead of directly for p.

The detection of singularity in M (given by (3.2) or (3.15)), as described in Section 3.2, required a significant amount of computation. However, by using the Schur complement formed in the above calculations, the amount of computation can be significantly reduced. This reduction is based on the observation that the new S is nonsingular if and only if M is nonsingular. Thus, singularity in M can be detected while forming the factorization of the new S.

In a null-space projection method the projected Hessian serves the dual purpose of solving for the search direction and verifying the necessary optimality condition that the projected Hessian be positive semi-definite. In a Lagrangian method, the search direction is determined without explicit use of the projected Hessian. However, when the method terminates, the definiteness of the projected Hessian must be determined in order to verify the optimality condition. The projected Hessian is not explicitly available in the Lagrangian method, but its definiteness can easily be determined from the factorization of the current K, i.e., the matrix in (3.34).

The solution of (3.34) involves the factorization of B into LDL^T, where L is lower triangular and D is block diagonal with blocks no larger than 2×2 (see Bunch and Kaufman, 1977, and Bunch and Parlett, 1971). If a matrix F is symmetric and a matrix J is nonsingular, then F and JFJ^T have the same inertia. Therefore, B and D have the same inertia. The special form of D makes its inertia (and hence, the inertia of B) easy to find. The inertia of the current K is determined by combining the inertias of S and B. Knowing the inertia of K, the inertia of the projected Hessian is readily available using the following relationship (Gould, 1984):

$$\text{In}(K) = \text{In}(Z^T H Z) + (t, t, 0),$$

where t is the number of rows of C.

The inertia of S is not usually determined. However, when the method ter-

minates, we can calculate the inertia of S in order to determine if the optimality condition is satisfied. If it is satisfied, the current iterate is a local minimum. If it is not satisfied, the current iterate is a constrained stationary point which is not a local minimum. From this constrained stationary point there is certainly a direction of negative curvature. However, no efficient means is available to calculate a direction of negative curvature in this case.

Chapter Four

A Method for Dense Quadratic Programs

The single-phase method in the previous chapter uses a Lagrangian method to solve the augmented system. The single-phase method for dense QP's described in this chapter, *QPSFA*, uses a projection method. (A pseudo-Fortran version of *QPSFA* is given in Table 2.) In order to test most aspects of a single-phase method it does not particularly matter what method is used to solve the augmented system, assuming that the problems are dense. A single-phase method for dense QP's, rather than for large-scale QP's, has been implemented for several practical reasons. First, more test problems can be run since it is computationally less expensive to test small dense problems, than large-scale problems. Second, much of the code necessary in *QPSFA* was readily available from *QPSOL*. Third, it made a comparison with *QPSOL* possible. Finally, we know of no generally available code for large-scale QP.

For purposes of exposition, the quadratic programming problem solved by *QPSFA* is assumed to be stated in the following form:

$$\text{QP2} \qquad \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad c^T x + \tfrac{1}{2} x^T H x$$

$$\text{subject to} \quad Ax \geq b.$$

The vector $r \equiv Ax - b$ will denote the residuals.

All constraints in QP2 are general constraints—there are no bound constraints. This simplification limits the description of updates to adding and deleting general constraints. (However, the implementation of *QPSFA* treats bounds and general constraints separately, since separate treatment is more efficient; see Gill *et al.*, 1982.)

The results of Chapter Three are stated for a problem of the form QP1. These results remain valid in this chapter since a problem of the form QP2 can be transformed into a problem of the form QP1 by adding slack variables. In a large-scale problem, there is no particular objection to the use of slack variables, since only the nonzero elements of the constraint matrix are stored. Since all of the elements of the constraint matrix are stored in a dense problem, it is preferable not to add slack variables. *QPSFA* does not explicitly use slack variables. Instead, *QPSFA* implicitly sets a slack variable equal to the corresponding residual, if the general constraint is satisfied, and equal to zero, if the general constraint is violated. This avoids storing and updating the slack variables, but implies that movement along the search direction may hit more than one zero-valued slack variable bound at a step of zero. Thus, the nondegeneracy assumption of Theorem 2 may not be satisfied. However, very rarely does this present a problem in practice (see Section 4.9).

The first section contains an outline of *QPSFA*. The next two sections describe the required factorizations and how they are updated. Calculation of the search direction from a non-optimal constrained stationary point is presented in Section 4.4. Section 4.5 contains a description of how *QPSFA* handles an indefinite Hessian. (*QPSFA* maintains a projected Hessian that has at most one nonpositive eigenvalue. In some instances in this thesis it may have been more complete to use the phrase "Hessian of unspecified definiteness" instead of "indefinite Hessian". However, this longer phrase has not been used in order to avoid wordiness.) Constraint deletion strategies are then described in Section 4.6. Resolution of singularity in the augmented system is discussed in Section 4.7. Infeasibility detection and convergence to an optimal point are discussed in Section 4.8. The last section compares *QPSFA* to the QP methods described in Section 2.6.

4.1. Outline of *QPSFA*

The main steps of *QPSFA* are:

1. **Initialise.** Determine which violated or exactly satisfied constraints are in the initial working set. Let the subscripts "w" and "I" denote a vector or matrix whose elements are associated with constraints in the working set and constraints not in the working set, respectively. Thus, the constraints in the working set determine the $t \times n$ matrix A_w ($t \leq n$), and the vectors b_w and r_w.
2. **Test for convergence.**
3. **Find p and μ.** Using a projection method, solve

$$K \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ r_w \end{pmatrix}, \qquad \text{where} \qquad K = \begin{pmatrix} H & A_w^T \\ A_w & \end{pmatrix}.$$

4. **Either delete a constraint or calculate the step length.**
   a. If a constraint is deleted, update K and return to step 2.
   b. If the step length is calculated, continue with step 5. (If $a_i^T p < 0$ for any violated constraint, α is zero. Otherwise, α is the step to the nearest satisfied constraint (not in the working set) along the search direction.)
5. **Update.** $x \leftarrow x + \alpha p$, $r_w \leftarrow (1 - \alpha) r_w$, and $r_I \leftarrow r_I + \alpha A_I p$.
   If $\alpha < 1$, add a constraint and update K.
   Return to step 2.

Note that α is zero whenever $a_i^T p$ is negative for any violated constraint. This implies that the residual of *every violated constraint is nondecreasing.*

4.2. Factorization of the working set and the projected Hessian

QPSFA is a projection method that uses equation (2.9) to solve the augmented system for the search direction and the multipliers. In order to use these equations,

representations of T, Z, Y, and R are needed. In this section we describe these matrices in more detail.

Let n_z denote $n - t$, the number or columns of Z. The columns of the $n \times n_z$ matrix Z form a basis for the subspace of A_W (i.e., $A_W Z = 0$). If t is zero, Z may be taken as the n-dimensional identity matrix.

The matrix Z may be defined from the TQ factorization of A_W. The TQ factorization of A_W is defined by

$$A_W Q = (0 \quad T), \tag{4.1}$$

where Q is an $n \times n$ matrix, and T is an $t \times t$ "reverse" triangular matrix such that $T_{ij} = 0$ for $i + j \leq t$. When Q is orthonormal, the TQ factorization is simply a permuted version of the standard LQ factorization. The reasons for favoring this form will be discussed in Section 4.3.2 when procedures for updating the matrix factorizations following a constraint deletion, are considered.

From (4.1) it follows that the *first* n_z columns of Q can be taken as the columns of the matrix Z. Denote the remaining columns of Q by Y (the columns of Y form a basis for the subspace of vectors spanned by the rows of A_W).

The Cholesky factorization of the projected Hessian is given by

$$Z^T H Z = R^T R. \tag{4.2}$$

QPSFA is based on the TQ and Cholesky factorizations. As mentioned at the beginning of this section, the matrices to be stored include, from (2.9), the $n \times n$ matrix Q, the t-dimensional reverse triangular matrix T, and the n_z-dimensional upper triangular matrix R. The matrix Q is conceptually partitioned into two submatrices — Z, the first n_z columns, and Y, the last t columns, i.e.

$$Q = (Z \quad Y).$$

Changes in the working set will cause changes in these four matrices. From (4.1) we see that any transformations applied to the columns of Y will also be applied to the columns of T; from (4.2) it follows that any transformations applied to the columns of Z will also be applied to the columns of R.

4.3. Updating the factorizations

Unless the correct active set is known a *priori*, the working set must be modified during the execution of an active-set method, by adding and deleting constraints. Because of the simple nature of these changes, it is possible to *update* the necessary matrix factorizations to correspond with the new working set. In the remainder of this section, we consider how to update the TQ factorization (4.1) and the Cholesky factorization (4.2) following a single change in the working set. If several constraints are to be added or deleted, the procedures are repeated as necessary.

We assume throughout this section that Q is orthonormal. The discussion of updates will assume a general familiarity with the properties of *plane rotations*. Sequences of plane rotations are used to introduce zeros into appropriate positions of a vector or matrix, and have exceptional properties of numerical stability (see, e.g., Wilkinson, 1965, pp. 47–48).

We shall illustrate each modification process using sequences of simple diagrams, following the conventions of Cox (1981) to show the effects of the plane rotations. Each diagram depicts the changes resulting from one plane rotation. The following symbols are used:

- x denotes a non-zero element that is not altered;
- m denotes a non-zero element that is *modified*;
- f denotes a previously zero element that is *filled in*;
- 0 denotes a previously non-zero element that is annihilated; and
- . (or blank) denotes a zero element that is unaltered.

In the algebraic representation of the updates, barred quantities will represent the "new" values.

4.3.1. Adding a general constraint. When a general constraint is added to the working set, its index can simply be placed at the end of the list of indices of general constraints in the working set. Therefore, without loss of generality we shall assume that the new constraint is added as the *last* row of A_W. The row dimension of A_W and the dimension of T will thus *increase* by one, and the column dimension of Z will *decrease* by one. Let a^T denote the new row of A_W. Let w^T denote the vector a^TQ, and partition w^T as $(w_Z^T \quad w_Y^T)$, so that w_Z^T consists of the first n_Z components of w^T. From (4.1), it follows that

$$\bar{A}_W Q = \begin{pmatrix} A_W \\ a^T \end{pmatrix} Q = \begin{pmatrix} 0 & T \\ a^TQ \end{pmatrix} = \begin{pmatrix} 0 & T \\ w_Z^T & w_Y^T \end{pmatrix}.$$

We see that a new matrix $\bar{Q}$ can be obtained by applying a sequence of plane rotations on the *right* of Q to transform the vector w_Z^T to suitable form; the transformed matrix Q then becomes $\bar{Q}$. The sequence of rotations take linear combinations of the elements of w_Z^T to reduce it to a multiple (say, γ) of e_Z^T, where e_Z denotes the n_Z-th coordinate vector. The rotations are constructed to alter pairs of components in the order $(1,2)$, $(2,3)$, ..., $(n_Z - 1, n_Z)$, as indicated in the following diagrams, which depict the vector w_Z^T as it is reduced to γe_Z^T:

$$(\times\ \times\ \times\ \times\ \times) \rightarrow (\circ\ \boxtimes\ \times\ \times\ \times) \rightarrow (\cdot\ \circ\ \boxtimes\ \times\ \times) \rightarrow (\cdot\ \cdot\ \circ\ \boxtimes\ \times) \rightarrow (\cdot\ \cdot\ \cdot\ \circ\ \boxtimes).$$

The effect of these transformations can be expressed algebraically as

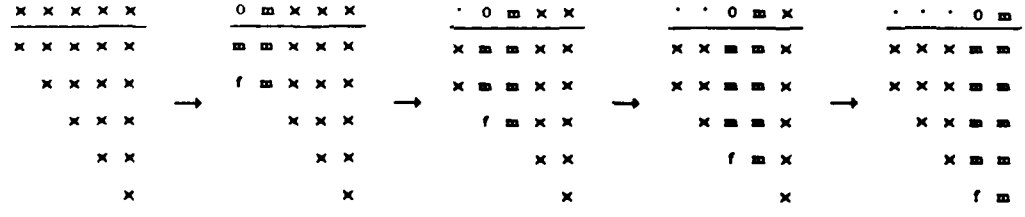
$$\bar{A}_W Q \begin{pmatrix} P & 0 \\ 0 & I \end{pmatrix} = \bar{A}_W \bar{Q} = \begin{pmatrix} 0 & 0 & T \\ 0 & \gamma & w_Y^T \end{pmatrix} = \begin{pmatrix} 0 & \bar{T} \end{pmatrix}.$$

By construction, the rotations in P affect only the first n_z columns of Q, so that the last t columns of $\bar{Q}$ are identical to those of Q, and the first n_z columns of $\bar{Q}$ are linear combinations of the first n_z columns of Q. Hence,

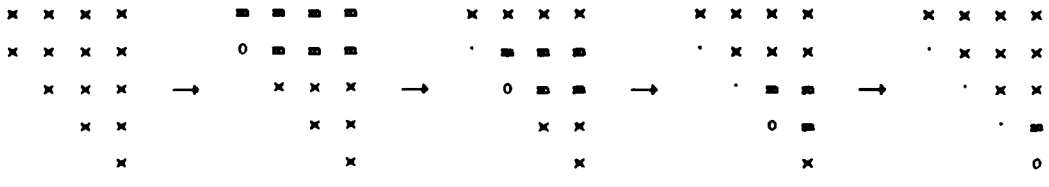
$$ZP = (\bar{Z} \quad y),$$

where y, the transformed last column of Z, becomes the first column of $\bar{Y}$.

The plane rotations applied to Z also transform the Cholesky factor R of the projected Hessian. The chosen order of the rotations in P means that each successive rotation has the effect of introducing a subdiagonal element into the upper-triangular matrix R, as shown in the following sequence of diagrams. For clarity, we again show the vector w_z^T at the top as it is reduced to $(0 \quad \gamma)^T$; the matrices underneath represent the transformed version of R.



Since the last column of the matrix ZP is not part of $\bar{Z}$, the last column of RP can be discarded. The remaining matrix is then restored to upper-triangular form by a forward sweep of *row* rotations (say, the matrix $\bar{P}$), which is applied on the *left* to eliminate the subdiagonal elements, as shown in the following diagrams.



Let $\tilde{R}$ denote the matrix RP with its last column deleted; then we have

$$\bar{P}\tilde{R} = \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix},$$

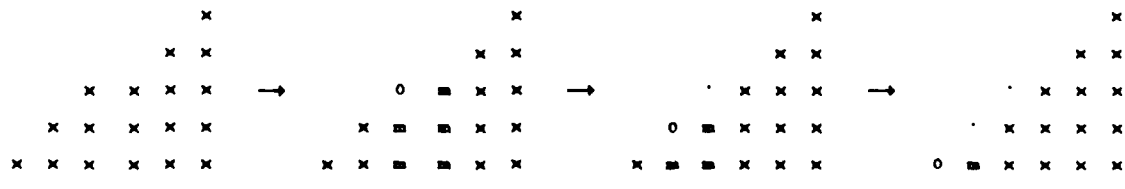
where $\bar{R}$ is upper-triangular. Note that the rotations in $\bar{P}$ affect $\bar{R}$, but not $\bar{Q}$ or $\bar{T}$.

The number of multiplications associated with adding a general constraint includes the following (where only the highest-order term is given): n^2 to form $A_w^T Q$; $3n_z^2$ for the two sweeps of rotations applied to R; and $3nn_z$ to transform the appropriate columns of Q. (We assume the three-multiplication form of a plane rotation; see Gill and Murray, 1974.)

4.3.2. Deleting a general constraint. When a general constraint (say, the i-th) is deleted from the working set, the row dimension of A_w and the dimension of T are decreased by one, and the column dimension of Z is *increased* by one. From (4.1), we have

$$\bar{A}_w Q = (0 \quad S),$$

where S is an $(t-1) \times t$ matrix such that rows 1 through $i-1$ are in reverse triangular form, and the remaining rows have one extra element above the reverse diagonal. In order to reduce the last $t-1$ columns of S to the desired reverse triangular form of $\bar{T}$, a backward sweep of plane rotations (say, P) is applied on the right, to take linear combinations of columns $(t-i+1, t-i)$, ..., $(2,1)$. The effect of these rotations is shown in the following diagrams for the case $t = 6$ and $i = 3$:



This transformation may be expressed as

$$\bar{A}_w Q \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} = \bar{A}_w \bar{Q} = (0 \quad \bar{T}). \tag{4.3}$$

The first n_z columns of Q are not affected by the rotations in P, and hence the first n_z columns of $\bar{Z}$ are identical to the columns of Z. The matrix $\bar{Z}$ is given by

$$\bar{Z} = (Z \quad z), \tag{4.4}$$

where the new (last) column z of $\bar{Z}$ is a linear combination of the relevant columns of Y. (When $i = t$, no reduction at all is needed, and z is just the first column of Y.)

By using the sweep of plane rotations described above, it is now straightforward to prove the following lemma.

Lemma 6. *If the j-th general constraint is deleted from the working set,*

$$a_j^T z = \sigma t_j,$$

where z is the new column of $\bar{Z}$ (4.4), t_j is the j-th diagonal element of T, and σ is the product of $t - j$ sines.

Proof:

By applying the same plane rotations to $a_j^T Y$, that were used to find $\bar{T}$ (4.3), the vector $a_j^T Y$ will be transformed into the vector $a_j^T (z \quad \bar{Y})$. The element $a_j^T z$ will be the first component of this transformed vector.

A plane rotation is equal to an identity matrix except for rows and columns i and j. Assuming i is less than j, the (i,i), (i,j), (j,i) and (j,j) elements have the configuration

$$\begin{pmatrix} c & s \\ s & -c \end{pmatrix},$$

where $c^2 + s^2 = 1$ (hence, $c = \cos\theta$ and $s = \sin\theta$ for some θ).

The first $t-j$ components of the t-dimensional vector $a_j^T Y$ are zero. The first of the remaining j components is t_j. The first plane rotation will form a transformed vector that starts with $t-j-1$ zero components, followed by the product of a sine with t_j. Each successive plane rotation has a similar effect. This implies that after $t-j$ plane rotations have been applied to $a_j^T Y$, the first component, $a_j^T z$, will be the product of $t-j$ sines with t_j. ∎

Because of the form (4.4), the new projected Hessian matrix $\bar{Z}^T H \bar{Z}$ is given by

$$\bar{Z}^T H \bar{Z} = \bar{R}^T \bar{R} = \begin{pmatrix} R^T R & v \\ v^T & \theta \end{pmatrix}, \tag{4.5}$$

where $v = Z^T H z$ and $\theta = z^T H z$.

Let r denote the solution of $R^T r = v$; if $\bar{Z}^T H \bar{Z}$ is positive definite, the quantity $\theta - r^T r$ must be positive. In this case, only one further step of the row-wise Cholesky factorization is needed to compute the new Cholesky factor $\bar{R}$, which is given by

$$\bar{R} = \begin{pmatrix} R & r \\ 0 & \rho \end{pmatrix},$$

where $\rho^2 = \theta - r^T r$. If the matrix $\bar{Z}^T H \bar{Z}$ is not positive definite, the Cholesky factorization may be undefined or ill-conditioned, and other techniques should be used to modify the factorization without excessive additional computation or loss of numerical stability (see Section 4.5.)

The number of multiplications associated with deleting the i-th general constraint includes the following (where only the highest-order term is given): $\frac{3}{2}(t-i)^2$ to operate on T; $3n(t-i)$ to transform Q; n^2 to form Hz; nn_z to form $Z^T Hz$; and $\frac{1}{2}n_z^2$ to compute the additional row of the Cholesky factor.

The justification for using the TQ factorization arises from this part of an active-set method. From a theoretical viewpoint, $\bar{Z}$ would remain an orthogonal basis for the null space of $\bar{A}_w$ regardless of the position in which the new column appeared. However, in order to update the Cholesky factors efficiently, the new column must appear *after* the columns of Z (otherwise, (4.5) would not hold). The TQ factorization has an implementation advantage because the new column of $\bar{Z}$ automatically appears in the correct position after deletion of a constraint from the working set. With other alternatives, the housekeeping associated with the update of R is more complicated. For example, in an implementation based on the LQ factorization, the new column might be moved to the end of Z, or a list could be maintained of the locations of the columns of Z; another alternative is to store the columns of Z in reverse order (see Gill and Murray, 1977).

4.4. Search direction calculation from a stationary point

If the current iterate is a non-optimal constrained stationary point and the residuals of the working set are zero, the calculation of the search direction can be simplified. At a constrained stationary point, we know that

$$Z^T g = 0. \tag{4.6}$$

The point is non-optimal, so a constraint with a negative multiplier will be deleted from the working set. The updated matrix Z, denoted by $\bar{Z}$, is given by

$$\bar{Z} = (Z \quad z). \tag{4.7}$$

From (4.6) we know that g will be orthogonal to all columns of $\bar{Z}$ except the last one. Hence,

$$\bar{Z}^T g = (z^T g) e_L. \tag{4.8}$$

Referring to equation (2.9), note that p_Y is zero, since r is zero. Using the Cholesky factorization of the updated projected Hessian, and equations (4.8) and (2.9*b*) gives

$$R^T R p_Z = -\gamma e_L. \tag{4.9}$$

The first step in solving (4.9) involves the lower-triangular matrix R^T. Because of the special form of the right-hand side of (4.9), the result is a multiple of e_L, and hence the vector p_Z is the solution of

$$R p_Z = -\frac{\gamma}{r_Z} e_L, \tag{4.10}$$

where r_Z is the last diagonal element of R.

In *QPSFA*, a search direction must be calculated even if the residuals are nonzero. Equation (4.10) can be used only if the residuals are zero. In a *feasible-point* active-set method that *only deletes constraints if the projected gradient is zero*, it can be shown that p_Z *is always the solution of* an *upper-triangular system of the form* (4.10) (see Gill *et al.*, 1981). This simplification in computing the search direction will frequently apply to the single-phase method. Also, it will be used to define a search direction when the projected Hessian is not positive definite.

4.5. Indefinite QP

4.5.1. Maintaining a positive definite projected Hessian. At a strong local minimum, the projected Hessian is positive definite. If the projected Hessian is not positive definite, a constrained stationary point is not a local minimum and the direction defined by (2.9*d*) is not necessarily a descent direction. Thus, it seems reasonable that the projected Hessian of our working set should be as positive definite as possible. *QPSFA* maintains a projected Hessian that has at most one nonpositive eigenvalue. With this restriction on the projected Hessian, it is possible to compute a feasible direction of descent for the quadratic objective function or a feasible non-ascending direction of negative curvature whenever the working set residuals are zero. In doing so, it is desirable to retain the maximum amount of information in the present (indefinite) projected Hessian, in order to preserve the efficiencies associated with quadratic programming; therefore, standard techniques that *alter* an indefinite matrix (see Gill, Murray and Wright, 1981) are not suitable. However, there is a danger of substantial numerical instability if care is not exercised in updating a factorization of an indefinite matrix.

QPSFA will solve an indefinite QP, provided the initial projected Hessian is positive definite. Under these circumstances, the only way in which the projected Hessian can become indefinite is when a single constraint is *deleted* from the working set; the effect of this change on the Cholesky factors of the new projected Hessian is that R is augmented by a single column and the new last diagonal element of R, denoted by r_L, is zero or undefined (being the square root of a negative number). However, the definition (4.10) of the search direction from a non-optimal constrained stationary point, is such that the last diagonal element of R affects only the *scaling* of p_z and not its direction. Consequently, without any loss of generality, we may use the modulus of the operand when computing the square root for the last diagonal

element of R. In this special situation, the search direction defined by (2.9*d*) is a direction of negative curvature. The last diagonal element of the modified R is denoted by r_z, and is given by $r_z^2 = -r_t^2$.

By altering R as described above, we are conceptually making a rank-one change to H. To see this, assume that $Z^T H Z$ is positive definite. If a variable is released from its bound and the new projected Hessian, $\bar{Z}^T H \bar{Z}$, is indefinite (one nonpositive eigenvalue), we could modify H to form $\bar{H}$, giving

$$\bar{H} = H + \sigma z z^T, \tag{4.11}$$

where $z^T \bar{Z} = e_z^T$ (last element of e_z^T is one), and σ is a positive constant. In order to make the above alteration to R, define σ, by $\sigma \equiv -2r_t^2$. This implies that the modified $R^T R$, given by

$$R^T R = \bar{Z}^T \bar{H} \bar{Z} = \bar{Z}^T H \bar{Z} + \sigma e_z e_z^T, \tag{4.12}$$

is positive definite. We never need to explicitly find $\bar{H}$. Instead, modify R by making its last diagonal element positive rather than the square root of a negative number. If the modification is not needed, define $\bar{H} \equiv H$. This modification implies that $Z^T \bar{H} Z$ *will always be positive definite.*

The following lemma will prove that if the search direction is calculated using the modified R, it will be a direction of negative curvature.

Lemma 7. *Suppose that r_t^2 is negative when a constraint is deleted from the working set. If the modified R given in (4.12) is used, the search direction (4.10) will be a direction of negative curvature.*

Proof:

Pre- and post-multiplying (4.12) by p_z^T and p_z, respectively, and rearranging terms gives

$$p_z^T \bar{Z}^T H \bar{Z} p_z = p_z^T R^T R p_z - \sigma (p_z^T e_z)^2. \tag{4.13}$$

In order to show that the search direction is a direction of negative curvature, the left side of (4.13) will be shown to be negative.

The terms on the right side of (4.13) will now be expressed in equivalent forms. Consider the first term of the right side of (4.13). Using (4.10) gives

$$p_z^T R^T R p_z = \frac{\gamma^2}{r_z^2}. \tag{4.14}$$

Now consider the last term of (4.13). Denote the last element of p_z by p_L. Using (4.10), p_L is given by

$$p_L = -\frac{\gamma}{r_z^2}. \tag{4.15}$$

Using the definition of σ, equation (4.15), and the relationship $r_z^2 = -r_L^2$, the last term of (4.13) can be written as

$$\sigma(p_z^T e_z)^2 = \sigma p_L^2 = -2r_L^2 \frac{\gamma^2}{r_z^4} = 2\frac{\gamma^2}{r_z^2}. \tag{4.16}$$

Using (4.16) and (4.14), equation (4.13) can be rewritten as

$$p_z^T \bar{Z}^T H \bar{Z} p_z = \frac{\gamma^2}{r_z^2} - 2\frac{\gamma^2}{r_z^2}. \tag{4.17}$$

Thus, $p_z^T \bar{Z}^T H \bar{Z} p_z$ is negative, as required. ∎

Suppose the current iterate is a constrained stationary point, the projected Hessian is positive definite, and the smallest multiplier, λ_j, has a value of zero. If the j-th constraint is deleted and the projected Hessian is now indefinite, how do we compute a direction of negative curvature? In this case, we shall show that (4.10) does not define a direction of negative curvature.

Lemma 8. *If* $\|Z^T g\| = 0$, *and the constraint* $a_j^T x \geq b_j$ *with multiplier* λ_j, *is deleted from the working set, then* $\|\bar{Z}^T g\| = z^T a_j \lambda_j$.

Proof: Define the new constraint matrix, $\bar{A}_W$, by

$$\begin{pmatrix} \bar{A}_W \\ a_j^T \end{pmatrix} = A_W. \tag{4.18}$$

Since the projected gradient is the zero vector,

$$g = A_W^T \lambda. \tag{4.19}$$

Express the new projected gradient by

$$\bar{Z}^T g = \begin{pmatrix} Z^T g \\ \\ z^T g \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ z^T g \end{pmatrix}. \tag{4.20}$$

Using (4.18) and (4.19), express $z^T g$ by

$$z^T g = z^T A_W^T \lambda = z^T \begin{pmatrix} \bar{A}_W^T & a_j \end{pmatrix} \lambda = z^T a_j \lambda_j, \tag{4.21}$$

since $\bar{A}_W z = 0$. ∎

If the multiplier of the deleted constraint is zero, Lemma 6 implies that γ (4.9) is zero also. This implies that the search direction is the zero vector. Thus, the above strategy to find a direction of negative curvature has failed.

In order to compute a direction of negative curvature when the multiplier of the deleted constraint is zero and the new projected Hessian is indefinite, we set γ (4.9) equal to one. Thus, γ has a nonzero value, so Lemma 7 proves that the search direction is a direction of negative curvature.

When we give γ a nonzero value, we are conceptually altering c. The new vector c and associated gradient are defined by $\bar{c} \equiv c + a_j$, and $\hat{g} \equiv g + a_j$. In this event, the new projected gradient can be expressed by

$$Z^T\hat{g} = Z^Tg + Z^Ta_j = Z^Ta_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ z^Ta_j \end{pmatrix}.$$

From Lemma 6, z^Ta_j is nonzero. Thus, γ (4.9) is nonzero.

In *QPSFA*, the vector Z^Tg is used so c is only conceptually being modified. In a large-scale implementation, the vector Z^Tg would not be available. In this case, we could use $\hat{g}$ instead of g in the augmented system.

In practice, when z^Tg is small, its sign is suspect due to roundoff error. Thus, it is better to determine the sign of p from the requirement $a_j^Tp > 0$.

Once a direction of negative curvature is computed, a constraint must be added during the next iteration (otherwise the objective function is unbounded below). It can be shown that the addition of a new constraint to the working set cannot increase the number of negative eigenvalues in the projected Hessian. Suppose that such a constraint "exchange" takes place, and that the last diagonal element of R was previously altered as described above in order to avoid taking the square root of a negative number. It can be shown that the arbitrary value resulting from the constraint deletion does not propagate to any other elements, i.e. the factors of the "exchanged" projected Hessian will still have a single arbitrary last diagonal element. This result implies that when a constraint exchange results in a positive-definite projected Hessian, the last diagonal element of the triangular factor must be recomputed.

There might appear to be a danger of numerical instability in allowing an indefinite projected Hessian. Certainly the occurrence of an undefined quantity

during the calculation of R implies that the usual bound on growth in magnitude in the elements of R from the positive-definite case (see Gill, Murray and Wright, 1981) does not apply. However, after a sequence of constraint exchanges, it is possible only for the *last row* of R to be "contaminated" by growth. It is possible to monitor the error in the last row and if need be it can be recomputed as soon as a positive-definite projected Hessian is obtained.

This result justifies tolerating a very limited form of indefiniteness, as described above. However, the overall viewpoint of the algorithm is that the projected matrix should be kept "as positive definite as possible". Therefore, once the projected Hessian is indefinite, *no further constraints are deleted* until enough constraints have been added so that the projected Hessian has become positive definite.

4.5.2. Initial projected Hessian. The treatment of indefiniteness in the projected Hessian, described in the previous section, is based on the assumption that the *initial* projected Hessian is positive definite. If H is indefinite, the initial projected Hessian (based on the initial working set) *will not necessarily* be positive definite. This section will describe the strategy used to determine an initial projected Hessian that is positive definite. For details on how this strategy was developed, see Gill *et al.*, 1985. The strategy is based on the observation that the projected Hessian matrix will be positive definite if enough constraints are included in the initial working set. (The null matrix is positive definite by definition, corresponding to the case when A_W contains n constraints.) This suggests somehow *adding artificial constraints to the working set* (that are deleted later) to make the projected Hessian positive definite.

One strategy is to add as many general constraints to the working set as necessary to create a temporary vertex, thus ensuring that the initial projected Hessian will be positive definite. This strategy is particularly attractive because it requires

no further work to update the TQ factorization or to compute the projected Hessian.

The computation proceeds as follows. At the beginning of the QP phase, the working set A_w and its TQ factorization (4.1) are available. The matrix Z^THZ is formed, and the Cholesky procedure *with symmetric interchanges* is initiated. Recall that the Cholesky procedure without interchanges will break down if the matrix is not positive definite. However, by performing interchanges (such that the column with largest positive diagonal element is processed next at each step), we can identify the largest possible positive-definite principal minor.

In algebraic terms, assume that a permutation matrix P has been chosen so that the upper left submatrix of P^TZ^THZP is positive definite. In effect, the columns of ZP are partitioned as $ZP = (Z_1 \;\; Z_2)$, such that $Z_1^THZ_1$ is positive definite, i.e.,

$$Z_1^THZ_1 = R_z^TR_z.$$

A working set for which Z_1 defines the null space can be obtained by including *the rows of* Z_2^T as temporary *general* constraints. After P is determined (by the Cholesky procedure), the columns of Z are reordered (i.e., Z is replaced by ZP); note that the properties of Z as a basis for the null space of A_w are unaffected by its column ordering. The minimization of the quadratic function then proceeds within the subspace defined by Z_1.

We discuss here only the case when Q is orthogonal. (For details about the case when Q is a product of stabilized elementary transformations, see Gill *et al.*, 1985.) The temporarily augmented working set is given by

$$\bar{A}_w = \begin{pmatrix} A_w \\ Z_2^T \end{pmatrix}, \tag{4.22}$$

so that p will satisfy $A_wp = 0$ and $Z_2^Tp = 0$. By definition of the TQ factorization,

$\bar{A}_W$ *automatically* satisfies the following:

$$\bar{A}_W Q = \begin{pmatrix} A_W \\ Z_2^T \end{pmatrix} Q = \begin{pmatrix} A_W \\ Z_2^T \end{pmatrix} (Z_1 \quad Z_2 \quad Y) = (0 \quad \bar{T}),$$

where

$$\bar{T} = \begin{pmatrix} 0 & T \\ I & 0 \end{pmatrix},$$

and hence the TQ factorization of (4.22) is free.

The implementation of this procedure involves several subtle points. The matrix Z_2 need *not* be kept fixed at its initial value, since the role of the extra constraints is purely to define an appropriate null space; the TQ factorization can therefore be updated in the normal fashion as the iterations of the quadratic programming method proceed. No work is required to "delete" the temporary constraints associated with Z_2 when $Z_1^T g = 0$, since this simply involves repartitioning Q. When deciding which constraint to delete, the multiplier vector associated with the rows of Z_2^T is given by $Z_2^T g$, and the multipliers corresponding to the rows of the "true" working set A_W are the least-squares multipliers that would be obtained if the temporary constraints were not present (see Gill *et al.*, 1985).

QPSFA starts with a positive definite projected Hessian and keeps it as positive definite as possible. The single-phase method of Chapter Three is a more general method that does not impose this requirement. We could make the single-phase method coincide with *QPSFA* by starting it with a positive definite projected Hessian (a vertex if necessary) and using the same strategy that *QPSFA* uses to keep the projected Hessian as positive definite as possible. By doing this, the single-phase method and *QPSFA* would handle indefiniteness in the same way. However, to start with a positive definite projected Hessian, the single-phase method using the Lagrangian method to solve the augmented system, would add artificial bounds. The

single-phase method could not use the above *QPSFA* strategy of adding rows of Z as temporary general constraints, since Z is not available in a Lagrangian method.

4.6. Constraint deletion strategies

4.6.1. Introduction. In *QPSFA*, the constraint exchange accomplished when the SR finds a suitable constraint to delete, can be viewed as a constraint deletion and a constraint addition. Thus, constraints are added in two circumstances and deleted in two circumstances (see Section 4.9). The strategies used for adding constraints in both circumstances are identical and straightforward. If a step of unity can be taken, without hitting a constraint, no constraint is added. Otherwise, the constraint added is the closest one to the current iterate along the search direction. When deleting constraints there are several strategies possible. These strategies differ between the two circumstances and are not as straightforward. In the first circumstance, the SR determines if there is a suitable constraint to delete (see Section 4.6.4). If there is more than one suitable constraint, *QPSFA* must determine which one should be deleted (see Section 4.6.5). It must also decide if more than one constraint should be deleted (see Section 4.6.6). In the second circumstance, the constraint with the smallest negative multiplier is deleted. If more than one multiplier is negative, the algorithm must decide if more than one constraint should be deleted (see Section 4.6.6). The algorithm allows constraints to be deleted before the current iterate is a constrained stationary point (see Section 4.6.2). Also, even if the the current iterate is infeasible, constraint deletions can be made (see Section 4.6.3). *QPSFA* maintains a projected Hessian with at most one nonpositive eigenvalue. If the projected Hessian is indefinite (one nonpositive eigenvalue) and a constraint is deleted, the new projected Hessian might have more than one nonpositive eigenvalue. Thus, the algorithm requires that the projected Hessian be

positive definite before departing the inner repeat loop and deleting a constraint on the basis of a negative multiplier.

Some of the constraint deletion strategies discussed in this section do not affect the convergence of *QPSFA*. However, they are included for the following reason. As constraints are added to the working set, the search direction becomes less dependent on the quadratic objective function. If enough constraints are added, the working set determines a vertex. If this happens, the search direction is the vector from the current iterate to the vertex, and is, therefore, completely independent of the objective function. This would make the algorithm similar to a two-phase method, where the first phase consists of finding a feasible point by calculating the search directions to successive vertices. By allowing more constraint deletions to occur, the working set will be less likely to determine a vertex. Thus, the search direction will be more dependent on the objective function than if the additional deletions did not occur. These additional deletion strategies are described in Sections 4.6.2 and 4.6.3.

4.6.2. Constraint deletion before finding manifold minimum. Considerable effort has been made in many branches of mathematical programming to determine the "best" choice of constraints to be deleted (i.e. that which results in the least number of iterations on average). Experimental results are mixed, but intuitively a strategy based on maximizing the likely reduction in the objective function would seem to be superior (see Gill and Murray, 1979). Unfortunately, the effort to implement such a strategy by known techniques is prohibitively expensive for almost all types of problems (including QP), even assuming an optimistic view of the reduction in the number of iterations. Hence, many methods choose the constraint with the most negative multiplier as the constraint to delete.

Many active-set methods also adopt the strategy of waiting until the current

iterate is a minimum on a manifold before deleting any constraint. If there is more than one negative multiplier at the minimum on a manifold, deleting a constraint corresponding to any such multiplier will give convergence in a finite number of iterations. The active-set method presented here will adopt the strategy of deleting the constraint with the most negative multiplier, but may delete constraints at points other than constrained stationary points.

If a constraint with a negative multiplier is deleted, we would like the new search direction to be a feasible direction with respect to the deleted constraint and to be a descent direction. This is in fact true, whether or not the current iterate is a constrained stationary point.

Theorem 4. *Suppose that r_w (2.14) equals zero and Z^THZ is positive definite. Assume that when (2.14) is solved, the j-th multiplier is negative. If constraint j is deleted, and the new search direction is denoted by $\bar{p}$, then $a_j^T\bar{p}$ is positive and $g^T\bar{p}$ is negative.*

Proof: Z^THZ is positive definite. When a constraint is deleted from the working set, express $\bar{Z}$ by $\bar{Z} = (\,Z \quad z\,)$. If the new projected Hessian, $\bar{Z}^TH\bar{Z}$, is indefinite, R is modified. In this case, we are conceptually changing H to $\bar{H}$, such that $\bar{Z}^T\bar{H}\bar{Z}$ is positive definite (see Section 4.5.1).

Let $\bar{A}_w$ denote A_w with the j-th row omitted. Similarly, denote μ with the j-th multiplier omitted, by $\bar{\mu}$. Let the vectors p and μ solve the augmented system (2.14), rewritten as

$$\begin{pmatrix} H & \bar{A}_w^T & a_j \\ \bar{A}_w & & \\ a_j^T & & \end{pmatrix} \begin{pmatrix} p \\ -\bar{\mu} \\ -\mu_j \end{pmatrix} = \begin{pmatrix} -g \\ 0 \\ 0 \end{pmatrix}. \tag{4.23}$$

Now, delete the j-th constraint from A_w, and let $\bar{p}$ and $\bar{\mu}$ solve the augmented

system given by

$$\begin{pmatrix} \bar{H} & \bar{A}_w^T \\ \bar{A}_w & \end{pmatrix} \begin{pmatrix} \bar{p} \\ -\bar{\mu} \end{pmatrix} = \begin{pmatrix} -g \\ 0 \end{pmatrix}. \tag{4.24}$$

The first rows of (4.23) and (4.24), respectively, give

$$Hp - \bar{A}_w^T \tilde{\mu} - a_j \mu_j = -g, \tag{4.25}$$

and

$$\bar{H}\bar{p} - \bar{A}_w^T \bar{\mu} = -g. \tag{4.26}$$

If $\bar{Z}^T H \bar{Z}$ is indefinite, modify H to give $\bar{H}$, using equation (4.19). Adding $\sigma z z^T p$ to both sides of (4.25), gives

$$\bar{H}p - \bar{A}_w^T \tilde{\mu} - a_j \mu_j = -g + \sigma z z^T p. \tag{4.27}$$

Since the vector p can be written as $p = Zp_z$ and $z^T Z$ is zero, the product $z^T p$ is zero. Thus, (4.27) can be simplified to

$$\bar{H}p - \bar{A}_w^T \tilde{\mu} - a_j \mu_j = -g. \tag{4.28}$$

Let $\hat{p} = \bar{p} - p$ and $\hat{\mu} = \bar{\mu} - \tilde{\mu}$. Subtract (4.28) from (4.26), and premultiply by $\hat{p}^T$ to form

$$\hat{p}^T \bar{H} \hat{p} - \hat{p}^T \bar{A}_w^T \hat{\mu} + \hat{p}^T a_j \mu_j = 0. \tag{4.29}$$

Equations (4.23) and (4.24) imply that $\bar{A}_w \hat{p}$ equals zero. This implies that $\hat{p}$ can be written as $\hat{p} = \bar{Z}\hat{p}_z$ for some $\hat{p}_z$. Also, $a_j^T p$ equals zero, so (4.29) can be written as

$$\hat{p}_z^T (\bar{Z}^T \bar{H} \bar{Z}) \hat{p}_z + \bar{p}^T a_j \mu_j = 0. \tag{4.30}$$

$\bar{Z}^T \bar{H} \bar{Z}$ is positive definite and μ_j is negative so (4.30) implies that $a_j^T \bar{p}$ is positive.

Next, premultiply (4.26) by $\bar{p}^T$ giving

$$\bar{p}^T \bar{H} \bar{p} - \bar{p}^T \bar{A}_w^T \bar{\mu} = -\bar{p}^T g. \tag{4.31}$$

$\bar{A}_w\bar{p}$ equals zero and $\bar{p}$ can be written as $\bar{p} = \bar{Z}\bar{p}_z$. Along with (4.31), these equations imply that

$$\bar{p}_z^T(\bar{Z}^T\bar{H}\bar{Z})\bar{p}_z = -\bar{p}^Tg. \tag{4.32}$$

$\bar{Z}^T\bar{H}\bar{Z}$ is positive definite so (4.32) implies that $\bar{p}^Tg$ is negative. ∎

Theorem 4 states that deleting a constraint with a negative multiplier before the projected gradient vanishes will result in a new search direction that is a descent direction and a feasible direction with respect to the deleted constraint. We shall now develop the specific strategy of deleting constraints before the projected gradient vanishes. (For other strategies see Gill and Murray, 1974, and Gould, 1982.)

Suppose that the j-th constraint has the most negative multiplier. Two possible extreme strategies for deleting the j-th constraint follow:

(*i*) Delete the j-th constraint only if the projected gradient is zero.

(*ii*) Delete the j-th constraint whenever its multiplier, μ_j, is negative.

If the projected gradient is near zero and μ_j is negative and has a "large" magnitude, it seems likely the objective function could be reduced more by deleting the j-th constraint. If the projected gradient is "large" and μ_j has a "small" magnitude, it seems unlikely that deleting the j-th constraint would result in a significant reduction in the objective function over that achieved by not deleting the j-th constraint. Moreover, since a large step usually implies additional constraints become active, it may be the multiplier of the j-th constraint changes sign in subsequent iterations. Thus, a strategy that uses the relative magnitudes of μ_j and the projected gradient, would appear to be better than either (*i*) or (*ii*) separately.

Let x denote the current iterate, $\bar{x}$ denote the minimum of the current manifold, and $\hat{x}$ denote the minimum of the current manifold with constraint j deleted. Denote the value of the objective function at x by $F(x)$. Let g denote the gradient at x, and

$\bar{g}$ denote the gradient at $\bar{x}$. Finally, define the following search directions: $p \equiv \bar{x} - x$, $\hat{p} \equiv \hat{x} - x$, and $\bar{p} \equiv \hat{x} - \bar{x}$.

Following are expressions for the differences in objective values between x and $\bar{x}$, and between $\bar{x}$ and $\hat{x}$:

$$F(\bar{x}) - F(x) = \tfrac{1}{2} p^T g, \tag{4.33}$$

and

$$F(\hat{x}) - F(\bar{x}) = \tfrac{1}{2} \bar{p}^T \bar{g}. \tag{4.34}$$

The term $p^T g$ is readily available, but $\bar{p}^T \bar{g}$ is not. We shall now derive a different expression for $\bar{p}^T \bar{g}$ which can be computed easily. Since $\bar{g}^T$ equals $\mu^T A_w$, $A_w p$ is zero, and $A_w \hat{p}$ equals $a_j^T \hat{p}$ times the j-th unit vector, the product $\bar{g}^T \bar{p}$ can be written as

$$\bar{g}^T \bar{p} = \mu^T A_w \bar{p} = \mu_j a_j^T \hat{p}. \tag{4.35}$$

Let Z correspond to the current manifold and $\bar{Z}$ to the current manifold excluding the j-th constraint. Define the vector z by $\bar{Z} = (\, Z \quad z\,)$. The vector $\hat{p}$ can be written as $\bar{Z}\hat{p}_z$ for some $\hat{p}_z$. Also, by the definition of Z, $a_j^T Z$ equals zero. Express $a_j^T \hat{p}$ by

$$a_j^T \hat{p} = a_j^T \bar{Z} \hat{p}_z = a_j^T z \tilde{p}, \tag{4.36}$$

where $\tilde{p}$ is the last component of $\hat{p}_z$.

Using (4.35), (4.36), and (4.9) in order, implies that $\bar{p}^T \bar{g}$ can be expressed as

$$\bar{p}^T \bar{g} = \mu_j a_j^T \hat{p} = \mu_j a_j^T z \tilde{p} = \mu_j \tilde{p} \sigma t_j = \delta \mu_j, \tag{4.37}$$

where δ is a positive constant.

The decrease in objective value from the current iterate to the minimum on the current manifold is given by $-\frac{1}{2} p^T g$. (See Fletcher, 1980, for the case where the

QP is positive definite and does not have any general constraints.) The decrease from the current iterate to the minimum on the current manifold excluding the j-th constraint is given by $-\frac{1}{2}p^Tg - \delta\mu_j$. Calculating δ for each constraint is computationally expensive. Also, even if δ were calculated, the above reductions may not be achieved since $\bar{x}$ and $\hat{x}$ may violate an inactive constraint. Thus, the following heuristic strategy will be used in *QPSFA*.

If constraint j has the most negative multiplier, it will be deleted if

$$\frac{p^Tg}{\mu_j} \leq \theta, \tag{4.38}$$

where θ is a positive parameter. Thus, if the projected gradient is small relative to the magnitude of the smallest multiplier and θ, the corresponding constraint will be deleted. In Chapter Five we report results for different values of θ. Note, that if θ is set equal to zero, this strategy is the same as (*i*). If θ is large enough, this strategy is the same as (*ii*).

In Section 4.5.1, the assumption was made that the current iterate was a non-optimal constrained stationary point. In this case, we can delete a constraint and find a direction of negative curvature, since $\bar{Z}^THZ$ is indefinite. *QPSFA* allows for deletions when *not* at a constrained stationary point. In this case, Theorem 4 has shown that the search direction is a *descent direction*. If $\bar{Z}^THZ$ is indefinite after deleting a constraint other than at a constrained stationary point, the search direction is *not* necessarily a direction of negative curvature, so the step length is limited to unity. If a constraint is not hit prior to a step of unity, $\bar{Z}$ is unchanged and $\bar{Z}^THZ$ remains indefinite. However, the *next* search direction *is* a direction of negative curvature along which the step length is not limited.

4.6.3. Constraint deletion at an infeasible point. In general, if a residual of any constraint in the working set is nonzero, and a constraint is deleted (even

one with a negative multiplier), there is no guarantee that the search direction will be a descent direction and move feasibly with respect to the deleted constraint. (Theorem 4 requires that the residuals in the working set equal zero.) Thus, the same constraint that was deleted could be added again before the current iterate changes, and cycling could result.

Note that the working set may not contain all of the general constraints that are violated at a given point. However, even though the current iterate is infeasible, constraints can be deleted as long as the working set residual r_w is zero.

Theorem 5. *If constraints with negative multipliers are deleted at points where* $r_w = 0$, *QPSFA will converge to a feasible point.*

Proof: We shall show that *QPSFA* cannot return to the same point, and thus will converge to a feasible point.

Theorem 4 states that the search direction is a feasible direction with respect to the deleted constraint. If $a_i^T p$ is positive for all violated general constraints, α is positive. This implies that each nonzero residual is decreased. Since the residual of each constraint is nonincreasing, the algorithm cannot return to the same point. If $a_i^T p$ is nonpositive for some violated general constraint, the constraint is added to the working set. This constraint has a nonzero residual, so the new r_w has a nonzero component. Since the new r_w is nonzero, the algorithm cannot delete another constraint until this new r_w is the zero vector again. Again, since the residual of each constraint is nonincreasing, the algorithm cannot return to the same point. ∎

If r_w is *not* zero and a constraint with a negative multiplier is deleted, there is no guarantee that the search direction is a descent direction *or* is a feasible direction with respect to the deleted constraint. However, if r_w is nonzero, a constraint with a negative multiplier is deleted, *and* the resulting $Z^T H Z$ is positive definite, the

search direction will be a feasible direction with respect to the deleted constraint. (Note that the definiteness of $\bar{Z}^T H \bar{Z}$ can be determined only after the constraint has been deleted.) The objective value at the minimum of the current working set with a constraint deleted is less than the objective value at the minimum of the current working set. However, both of these values may be greater than the objective value at the current iterate, since it is infeasible. Thus, in both cases, the search direction is not necessarily a descent direction. However, it still may be efficient to delete constraints in the above circumstance.

Theorem 6. *Assume that the j-th multiplier is negative when (2.14) is solved for p and μ. Let $\bar{p}$ and $\bar{Z}^T H \bar{Z}$ denote the search direction and projected Hessian when constraint j is deleted. If constraint j is deleted and $\bar{Z}^T H \bar{Z}$ is positive definite, then $a_j^T \bar{p}$ is positive.*

Let $\bar{A}_w$ denote A_w with the j-th row omitted. Similarly, let $\tilde{\mu}$ denote μ with the j-th multiplier omitted, and $\tilde{r}_w$ denote r_w with the j-th component omitted. Let the vectors p and μ solve the augmented system (2.14), rewritten as

$$\begin{pmatrix} H & \bar{A}_w^T & a_j \\ \bar{A}_w & & \\ a_j^T & & \end{pmatrix} \begin{pmatrix} p \\ -\tilde{\mu} \\ -\mu_j \end{pmatrix} = - \begin{pmatrix} g \\ \tilde{r}_w \\ r_j \end{pmatrix}. \tag{4.39}$$

Now, delete the j-th constraint from A_w, and let $\bar{p}$ and $\bar{\mu}$ solve the augmented system given by

$$\begin{pmatrix} H & \bar{A}_w^T \\ \bar{A}_w & \end{pmatrix} \begin{pmatrix} \bar{p} \\ -\bar{\mu} \end{pmatrix} = - \begin{pmatrix} g \\ \tilde{r}_w \end{pmatrix}. \tag{4.40}$$

The first rows of (4.39) and (4.40), respectively, give

$$Hp - \bar{A}_w^T \tilde{\mu} - a_j \mu_j = -g, \tag{4.41}$$

and

$$H\bar{p} - \bar{A}_w^T\bar{\mu} = -g. \tag{4.42}$$

Let $\hat{p} = \bar{p} - p$ and $\hat{\mu} = \bar{\mu} - \tilde{\mu}$. Subtract (4.41) from (4.42), and premultiply by $\hat{p}^T$ to form

$$\hat{p}^T H\hat{p} - \hat{p}^T\bar{A}_w^T\hat{\mu} + \hat{p}^T a_j\mu_j = 0. \tag{4.43}$$

Equations (4.39) and (4.40) imply that $\bar{A}_w\hat{p}$ equals zero. This implies that $\hat{p}$ can be written as $\hat{p} = \bar{Z}\hat{p}_z$ for some $\hat{p}_z$. Equation (4.43) can now be written as

$$\hat{p}_z^T(\bar{Z}^T H\bar{Z})\hat{p}_z + \bar{p}^T a_j\mu_j - p^T a_j\mu_j = 0. \tag{4.44}$$

The matrix $\bar{Z}^T H\bar{Z}$ is positive definite, μ_j is negative, and $p^T a_j$ is positive, so (4.44) implies that $a_j^T\bar{p}$ is positive. ∎

If a constraint with a negative multiplier is deleted and $\bar{Z}^T H\bar{Z}$ is indefinite, $a_j^T\bar{p}$ could be negative. This might cause the constraint to be immediately added. In this event, cycling could occur. Thus, the work to delete the constraint, update the factorizations (discovering that $\bar{Z}^T H\bar{Z}$ is indefinite), and then immediately add the constraint and regain the previous factorizations is work that would not have been required if the constraint had not been deleted. If H is negative definite, $\bar{Z}^T H\bar{Z}$ will always be indefinite. If H is indefinite, $\bar{Z}^T H\bar{Z}$ might be indefinite. The only way that $\bar{Z}^T H\bar{Z}$ will always be positive definite is if H is positive definite. Because of the potential for wasted effort, *this strategy will only be implemented when H is known to be positive definite.*

4.6.4. Choosing a constraint to delete using the SR. The SR will determine a suitable constraint to delete if β is positive for any fixed constraint (see Section 3.2.1). If β is positive for more than one constraint, *any* of the suitable constraints could be deleted. However, we would like to choose the "best" constraint to delete. It is more difficult to describe "best" in this context than when

discussing multipliers, since *QPSFA* searches simultaneously for an optimal *and* a feasible point. One good strategy would be to find a constraint to delete such that the new search direction could take a full step and satisfy all constraints in the working set. Such a constraint may not exist. Moreover, such a strategy would be prohibitively expensive to implement. However, a heuristic based on this approach may be implemented. This heuristic strategy chooses the constraint corresponding to the smallest positive β. Thus, the new search direction is "shortest" and the probability of hitting a constraint is "minimized".

Another strategy is to choose from the suitable, fixed constraints the one with the most negative multiplier. However, when the SR is used the current multipliers are not usually available. Thus, in order to test this strategy, (2.9*d*) must be solved each time the SR is used.

A third strategy is to choose from the suitable fixed constraints the one with the smallest ratio of the multiplier to the corresponding β. This combines the first two strategies by choosing a constraint with a small positive β and a large negative multiplier.

4.6.5. Multiple constraint deletions. Constraints are deleted in *QPSFA* in two circumstances. In the first circumstance, the SR finds a suitable constraint to delete to avoid singularity. In the second circumstance a constraint with a negative multiplier is deleted. If there are multiple permissible choices of constraints to delete in either circumstance, we may delete *more than one* constraint. If more than one constraint is deleted, the decision as to which is the "best" set to delete is very difficult.

It was shown in Section 4.6.2 that if a constraint with a negative multiplier is deleted when the working set residual is zero, the search direction is a descent direction that moves feasibly with respect to the deleted constraint. If more than

one constraint with a negative multiplier is deleted, the search direction is still a descent direction. However, it does *not* necessarily move in a feasible direction with respect to all of the deleted constraints. This implies that a deleted constraint could be hit and immediately added back to the working set. Thus, the multiple-deletion strategy could result in more computation time. If multiple deletions occur and the search direction *does* move feasibly with respect to all of the deleted constraints, the algorithm has calculated only one search direction and one set of multipliers, while "successfully" deleting several constraints. If the deletions had been made one at a time, the algorithm would have calculated a search direction and a set of multipliers for *each* deletion. Thus, the multiple-deletion strategy could result in less computation time. The effect that the multiple-deletion strategy has on the computation time depends on the particular problem.

An analogous dependence occurs when multiple deletions are made on the basis of the SR. We saw in Section 3.1 that if the SR finds a suitable constraint to delete, the search direction is a feasible direction with respect to the deleted constraint. If more than one suitable constraint is deleted, the search direction *does not necessarily* move in a feasible direction with respect to all of the deleted constraints. This implies that a deleted constraint could be hit and immediately added back to the working set. As in the previous paragraph, this implies that the effect on computation time is dependent on the particular problem.

Indefiniteness of the projected Hessian could prevent multiple deletions. If the SR is used and a suitable constraint is deleted, Z^THZ remains the same (see Lemma 7). If any additional suitable constraints are deleted, the projected Hessian could have more than one nonpositive eigenvalue. Thus, we may be restricted from deleting more than one constraint by the requirement that the projected Hessian has at most one nonpositive eigenvalue. Similarly, if a constraint with a negative

multiplier is deleted, the projected Hessian could become indefinite. If any additional constraints with negative multipliers are deleted, the projected Hessian could have more than one nonpositive eigenvalue. *QPSFA* maintains a projected Hessian with at most one nonpositive eigenvalue. If the projected Hessian is indefinite, constraints can not be deleted (exchanges using the SR are permissible) until the projected Hessian is positive definite again. Multiple constraint deletions might result in a projected Hessian with more than one nonpositive eigenvalue. Thus, if multiple constraint deletions are allowed, the definiteness of the projected Hessian must be determined after each deletion. Constraint deletions must not be allowed if they would result in a projected Hessian with more than one nonpositive eigenvalue. Thus, the work to determine if the new Z^THZ has more than one nonpositive eigenvalue, along with the work to return to the factorization of Z^THZ, is work that would not have been done if the multiple-deletion strategy were not used.

The determination of the "best" single constraint to delete is not performed because it is considered to be too computationally expensive. Thus, the heuristic rule of choosing the constraint with the largest multiplier or the smallest positive β is used. The determination of the "best" set of constraints to delete is certainly more difficult than choosing the best single constraint. Therefore, some heuristic rule must apparently be used again.

The primary objection to the multiple-deletion strategy is that a deleted constraint might immediately be hit and added back to the working set. Because of this objection and the other difficulties discussed in this section, we only consider the case of deleting a single constraint.

4.7. Resolution of singularity in *QPSFA*

Singularity in K can occur either when a variable is deleted or added to the working set. If singularity occurs when a variable is deleted, the new projected Hessian is singular. In this event (and when the projected Hessian becomes indefinite), we shall use the strategy of altering R—conceptually changing H to $\bar{H}$—that was described in Section 4.5.1.

If singularity occurs when a variable is added to the working set, the SR will be used. The SR derived in Section 3.1 assumes that the *Lagrangian method* will be used to solve a QP with general constraints that are equa*lities*. We shall now formulate a different derivation of the SR, one that is suitable for use with *QPSFA*—a *projection* algorithm with general constraints that are *inequalities*. Since the implementation of *QPSFA* treats bounds and general constraints separately, the following derivation of the SR also treats them separately. Thus, the derivation of the SR will use the matrix A_{FR}—the portion of A_W corresponding to the free variables.

This different derivation is done for two reasons. First, the matrix C contains all general constraints, so the calculation of β (3.12) only involves adding and deleting bounds. The implementation of *QPSFA* involves the TQ factorization of A_{FR}, which does not contain all general constraints. Thus, bounds and general constraints can be added and deleted. Second, if singularity occurs when a variable is added to the working set, the constraints are dependent, since $Z^T\bar{H}Z$ is positive definite. Since the singularity in the constraints does not depend on the Hessian, the SR can be derived without considering the Hessian. Thus, in *QPSFA* the detection and avoidance of singularity when a variable is added are simplified because the source of singularity is known to be in the constraints. Equation (3.12) will be equivalent to the new formula for β in the sense that deletion (addition) of a slack variable

from C is the same as adding (deleting) a general constraint to A_{FR}. First, a means to detect singularity when a variable is added to the working set will be described. Then, an equivalent formulation of the SR will be developed.

In Section 3.2 singularity in the new augmented system using a Lagrangian method could be detected only after updating the factorization or solving a linear system involving K. This could be computationally expensive. Since updates to the TQ factorization have already been described, the detection of singularity in the new T is straightforward. Singularity in the new T can occur only if a general constraint or a bound is *added* to the working set.

If a general constraint or a bound is added to the working set, the new T would be formed by applying a sequence of plane rotations to w_z^T (see Section 4.3.1). These rotations will transform w_z^T into the unit vector (with the last component one) times a nonzero number. If w_z^T is initially the zero vector, plane rotations *cannot* transform it into a vector with a nonzero final component. This implies that *if w_z^T is the zero vector, the new T would be singular.*

If a general constraint is being added, $w_z^T \equiv a_{FR}^T Z$, where a_{FR}^T corresponds to the free components of the general constraint that is being added. If a bound is being added, $w_z^T \equiv e_{FR}^T Z$, so w_z^T is the last row of Z (see Gill *et al.*, 1982). Thus, singularity in the new T can be detected before updating Q and T, and without the extra factorization or solve of a linear system required in the Lagrangian method.

Suppose the resulting T is singular if a constraint with index i were added to the working set. The SR tries to find a constraint, say with index j, that if deleted from the working set would ensure that (*a*) the new T would be nonsingular, and (*b*) the new search direction would be a feasible direction with respect to the deleted constraint.

In order to do this the SR solves the following system of equations for β:

$$\begin{pmatrix} A_{FR} & \ell \\ h^T & c \end{pmatrix} \begin{pmatrix} \bar{p} \\ \beta \end{pmatrix} = \begin{pmatrix} -r \\ -t \end{pmatrix}, \tag{4.45}$$

where r denotes r_w. Let $\bar{p}$ denote the new search direction, excluding the j-th component. The j-th component of the new search direction is denoted by β. If a bound is hit, then $h \equiv e_i$, and $t \equiv 0$ (forcing the i-th component of $\bar{p}$ to equal zero). If a general constraint is added to the working set, then h consists of the free components of the i-th row of A, and t is the corresponding (possibly nonzero) residual. (This forces the new search direction to satisfy the constraint indexed by i.) If a general constraint is deleted from the working set, $\ell \equiv -e_j$. If a variable is released from its bound, ℓ consists of the components of the j-th column of A corresponding to the general constraints in the working set. If a general constraint is hit, and a variable is released from its bound, c is the single element in the i-th row and j-th column of A; otherwise c is zero.

Let q be the solution of $(\,Z \quad Y\,)q = \bar{p}$. Rewrite (4.45) as

$$\begin{pmatrix} A_{FR} & \ell \\ h^T & c \end{pmatrix} \begin{pmatrix} Z & Y & \\ & & 1 \end{pmatrix} \begin{pmatrix} q \\ \beta \end{pmatrix} = -\begin{pmatrix} r \\ t \end{pmatrix}. \tag{4.46}$$

Now perform the indicated multiplication of the first two matrices in (4.46), and expand q into components relating to Z and Y. Equations (4.46) can then be written as

$$\begin{pmatrix} A_{FR}Z & AY & \ell \\ h^TZ & h^TY & c \end{pmatrix} \begin{pmatrix} q_Z \\ q_Y \\ \beta \end{pmatrix} = -\begin{pmatrix} r \\ t \end{pmatrix}. \tag{4.47}$$

Next, recalling that $A_{FR}Y = T$, $A_{FR}Z = 0$, and $h^TZ = 0$ (since the new T is singular), equation (4.47) can be written as

$$\begin{pmatrix} T & \ell \\ h^TY & c \end{pmatrix} \begin{pmatrix} q_Y \\ \beta \end{pmatrix} = -\begin{pmatrix} r \\ t \end{pmatrix}. \tag{4.48}$$

Hence

$$Tq_Y + \ell\beta = -r, \tag{4.49}$$

and

$$h^T Y q_Y + c\beta = t. \tag{4.50}$$

Let u be the solution of $h^T Y = u^T T$. Pre-multiplying (4.49) by u^T gives

$$h^T Y q_Y + u^T \ell\beta = -u^T r. \tag{4.51}$$

From (4.50) and (4.51) we get

$$-c\beta - t + u^T \ell\beta = -u^T r. \tag{4.52}$$

Hence

$$\beta = \frac{t - u^T r}{u^T \ell - c}. \tag{4.53}$$

Equation (4.53) determines a β for each constraint in the working set. Note that u need only be calculated *once* in order to calculate every β. If the denominator of (4.53) is not zero, β exists, which implies that the new T is guaranteed to be nonsingular. If β is positive (assuming j is on its lower bound), the new search direction would be a feasible direction with respect to the deleted constraint. Thus, the SR has found a permissible constraint to delete if any β has the proper sign.

In general, we are prohibited from deleting constraints when the projected Hessian is indefinite. However, the SR may have to be used, even though the projected Hessian is indefinite. We shall now examine how the constraint deletion performed when the SR is used, alters $Z^T \bar{H} Z$. A constraint deletion could increase the number of nonpositive eigenvalues of $Z^T \bar{H} Z$. However, the constraint deletions performed when the SR is used *will not change* $Z^T \bar{H} Z$. (By using the notation of Chapter 3, in which only bounds can be added to and deleted from the working set, the proof of the following lemma is simplified.)

Lemma 9. *If the SR is used, and $Z^T\bar{H}Z$ is positive definite, then the old Z is still a satisfactory basis for the null space of the new working set. Consequently, the new projected Hessian is positive definite.*

Proof: Use of the SR implies a variable (indexed by p) was hit that caused C to become singular, and a variable (indexed by q) was found that when released from its bound made the new C nonsingular. Since C became singular when the bound indexed by p was hit, Lemma 3 implies that $e_p^T Z = 0$. When the p-th bound is fixed and the q-th bound is released, the new C, denoted by $\bar{C}$, is formed by exchanging the q-th column for the p-th column, giving

$$\bar{C} = C + (a_q - a_p)e_p^T.$$

Thus, Z does not need to be altered since

$$\bar{C}Z = (C + (a_q - a_p)e_p^T)Z = CZ + (a_q - a_p)e_p^T Z = 0.$$

Similarly, the new H, denoted by $\bar{H}$, can be expressed as

$$\bar{H} = H + (h_q - h_p)e_p^T + e_p(h_q^T - h_p^T) - h_{dd}e_p e_p^T,$$

where h_q and h_p denote columns of H, and h_{dd} is the p-th element of $h_q - h_p$. When $Z^T\bar{H}Z$ is formed, we see that $Z^T\bar{H}Z$ is equal to Z^THZ. ∎

4.8. Infeasibility detection and convergence to an optimal point

In the single-phase Lagrangian method it is not possible to ascertain immediately if the problem is infeasible when the SR does not find a suitable variable to delete. In *QPSFA*, $Z^T\bar{H}Z$ is always positive definite. Thus, if the SR does not find a suitable variable to delete, Theorem 2 implies that the problem is infeasible.

Once a feasible point has been found, all future iterates remain feasible. Thus, the algorithm becomes an active-set feasible-point method. An outline of a convergence proof is given below. (In this proof, it is required that a minimum be found on each manifold.)

Theorem 7. *QPSFA will find an optimal point (or indicate unboundedness of the objective), given an initial feasible point.*

Proof: The inner repeat loop will find the minimum on a manifold in a finite number of iterations. If all multipliers are nonnegative, an optimal point has been found. Otherwise, a constraint with a negative multiplier is deleted. By the nondegeneracy assumption, the next step is positive. The search direction is a descent direction, so the objective value decreases. This implies that once the iterates leave a manifold, they cannot return to the same manifold. The objective value is monotonically decreasing, so the algorithm will find an optimal point, or a direction of negative curvature that does not hit any constraint. Furthermore, there are only a finite number of manifolds, so the algorithm will terminate in a finite number of iterations.

∎

Table 2

Pseudo-Fortran version of *QPSFA*

```
pd = ( H is positive definite )
feas = ( iterate satisfies all constraints )
repeat
        repeat
                statpt = ( projected gradient equals zero )
                wsnrm = ( norm of working set residuals equals zero )
                compute p
                compute second-order multipliers
                mlrptg = (a negative multiplier is large relative to
                            the gradient projected along p)
                if ( not ((statpt or mlrptg) and (pd or wsnrm)
                  and posdef and feas )) then
                        compute α
                        if ( constraint is hit ) then
                                if ( T is singular ) then
                                        if ( SR finds constraint to delete ) then
                                                exchange constraints
                                        else
                                                error = true
                                        end if
                                else
                                        add constraint
                                end if
                        end if
                        x̄_FR = x_FR + αp
                        posdef = ( projected Hessian is positive definite )
                        feas = ( iterate satisfies all constraints )
                end if
        until ( ( ( statpt or mlrptg ) and ( pd or wsnrm ) and posdef )
                or error )
        compute smallest multiplier
        optmul = ( smallest multiplier is greater than zero )
        if ( not ( optmul or error ) ) then
                delete constraint
                posdef = ( projected Hessian is positive definite )
        end if
until ( ( optmul and feas ) or error )
```

4.9. Comparison of *QPSFA* to alternative QP methods

Brief descriptions of three alternative QP methods were given in Section 2.6. We shall now compare *QPSFA* to these three methods. In terms of solving an EQP, the methods *QPSFA*, *QPSOL*, and *CS* are null-space projection methods which solve the augmented system using (2.9). The residual, r, and hence the p_Y term, is zero in *QPSOL*, but not in *QPSFA*. (Much of the computer code used by *QPSFA* is identical to the code of *QPSOL*.) *GI* is a range-space projection method which uses (2.11).

In addition to the fact that the iterates of *QPSFA* may be infeasible, the method is not dual feasible. In terms of describing how the method controls changes to the working set, this implies that the method is neither an active-set feasible-point method, nor a dual-feasible active-set method. Once a violated constraint becomes satisfied, the method does not allow the constraint to become violated again. However, once a multiplier becomes positive, the method does not attempt to keep it positive. In this sense, the method is more similar to an active-set feasible-point method than a dual-feasible active-set method. *QPSFA* is similar to *QPSOL* since it deletes constraints on the basis of negative multipliers, and adds constraints when they restrict the step to the minimum on the manifold. *QPSFA* and *GI* both add violated constraints, so in this sense they are similar. (However, *QPSFA* and *GI* use different criteria for adding these violated constraints. *GI* adds the constraint that is most violated at the current iterate. *QPSFA* adds the constraint that is most violated at a step of unity along the search direction.) *QPSFA* is also similar to *CS* in that it can be described as an exact penalty method, where the penalty function is given by

$$P(x,\rho,p) = F(x) + \begin{cases} \rho \sum_{i=1}^{t} |r_i|, & \text{if } a_i^T p > 0, \quad \text{for all } i \in \mathcal{V} \\ \infty, & \text{if } a_i^T p \leq 0, \quad \text{for any } i \in \mathcal{V} \end{cases},$$

where ρ is sufficiently large and $\mathcal{V}$ denotes the set of violated general constraints. Note that the summation involves only the t constraints *in the working set*, indicating that these violated constraints are the only ones that incur a penalty. Of course, one could not use such a penalty function in an algorithm, unless a method of determining a search direction that would not incur an infinite penalty were given. *QPSFA* determines such a search direction, so it can be viewed as using $P(x, \rho, p)$. Due to the infinite penalty, *QPSFA* takes steps of length zero until it finds an appropriate search direction. This is a different explanation of why the sum of infeasibilities in *QPSFA* is a decreasing function.

Each algorithm must indicate when the problem is infeasible. When the SR (in *QPSFA*) fails to find a constraint to delete, this implies that it has found an infeasible subproblem. *GI* also indicates infeasibility by finding an infeasible subproblem. Infeasibility is indicated in *QPSOL* when the first phase does not find a feasible point. With a sufficiently large ρ, infeasibility is indicated in *CS* when the solution of the unconstrained problem is infeasible with regard to the original problem.

QPSFA and *QPSOL* handle indefinite QP's in the same way—they allow at most one nonpositive eigenvalue in the projected Hessian. *GI* does not accommodate an indefinite Hessian. *CS* allows multiple negative eigenvalues in the projected Hessian.

The SR is unique to *QPSFA*. It is necessary in *QPSFA*, since the working set could become singular when a constraint is hit. This cannot occur in an active-set feasible-point method, such as *QPSOL*, because the search direction will not intersect any dependent constraint. To see this, assume that the vector a_j is the normal of a constraint that is not in the working set and that a_j is a linear combination of the rows of C. The search direction p satisfies $Cp = 0$, and it follows that $a_j^T p = 0$. Thus, p will not intersect the dependent constraint. In *QPSFA*, the search direction satisfies $Cp = -r$, so it could intersect a dependent constraint. *GI* handles

singularity in the working set when a constraint is added, by using the change in the existing multipliers to determine which constraint to delete.

The assumption of nondegeneracy is needed in feasible-point active-set methods, because of a problem that might occur when a constraint is deleted from the working set at a degenerate point. If the current iterate is a degenerate point, any positive step along the search direction may violate one of the dependent constraints that was not in the working set. Such a constraint must then be added to the working set. If x is not a vertex, a move can be made without deleting any more constraints from the working set. If x is a vertex, a constraint *must* be dropped from the working set. Thus, there is the chance that the sequence of working sets obtained by deleting and adding constraints may repeat itself after finitely many steps—a phenomenon known as *cycling*. Note that degeneracy is not a difficulty in itself; but when it is present, cycling is a possibility.

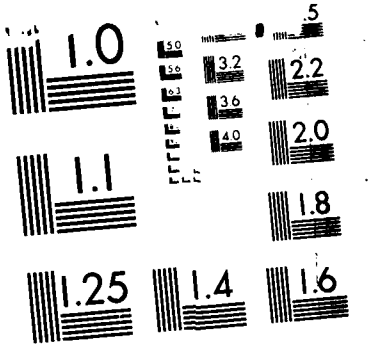
The resolution of degeneracy at a vertex, i.e. the computation of a search direction such that the objective function undergoes a strict decrease, is guaranteed if enough combinations of constraints are deleted from the working set. However, the resolution of degeneracy at a vertex is essentially a combinatorial problem whose solution may require a significant amount of computation. Fortunately, the occurrence of cycling is rare; when it does occur, simple heuristic strategies almost always succeed in breaking the deadlock. In order to establish convergence, nondegeneracy assumptions are made in *QPSFA*, *QPSOL*, and *CS*. In *GI*, a nondegeneracy assumption is not necessary since the Hessian is positive definite.

Of these four algorithms, *QPSFA* is the only one that allows constraints to be deleted when not at a minimum of a manifold. However, *QPSOL* and *CS* could be modified to allow such deletions. Such a strategy does not apply to *GI*, since deletions are made to maintain dual feasibility.

All four of these algorithms allow for the deletion of constraints when the current iterate is infeasible. However, in *QPSOL*, this occurs only in the first phase. The other three algorithms are single-phase methods, so such deletions must be permitted.

AD-A166 373 A SINGLE-PHASE METHOD FOR QUADRATIC PROGRAMMING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH S C HOYLE NOV 85 AFIT/CI/NR-86-3D 2/2

UNCLASSIFIED F/G 9/2 NL



Chapter Five

Numerical Results

This chapter presents computational results of the implementation of *QPSFA* (described in Chapter Four). The quadratic programs were randomly generated by a method that allows the user to specify the numerical properties of the problem (see Section 5.1). Several parameters must be input to *QPSFA* in order to specify the algorithm completely. The results obtained by using different combinations of these parameters are given in Section 5.2. The performances of *QPSFA* and *QPSOL* on identical problems are compared in Section 5.3.

A common way to compare algorithms is to use some measure of the amount of work performed (e.g., execution time, number of iterations). A large portion of the computational effort required in all QP methods is used to solve the augmented systems. The total amount of work is roughly proportional to the number of solves of the augmented system. (An augmented system must be solved each time a constraint is added to or deleted from the working set.) Thus, when comparing different parameter settings of *QPSFA*, and *QPSOL* to *QPSFA*, the comparisons will be based primarily upon the total number of solves of the augmented system.

5.1. Generation of random quadratic problems

The most common objective when testing a QP algorithm is to compare its efficiency with that of other algorithms. It is essential to be able to specify the numerical properties of the test problems. The method of generating dense QP's used in this thesis has this feature. (This method of generating QP's is described by Gould, 1984.)

With this method, the user is able to specify certain characteristics of the QP. The size of the problem, both in terms of the number of variables and the number of constraints, must be specified. The dimension of the null space at the solution is specified by inputing the desired number of active general constraints and bounds at the solution. The conditioning of the problem, both in terms of the objective function and the constraints, must be input. Indefinite QP's are formed by making some of the eigenvalues of the Hessian negative. Linear programs can be formed simply by specifying that the Hessian is the null matrix.

By varying these characteristics, a group of twenty "typical" problems was formed. (The number of variables was 20 or less and the number of general constraints was 50 or less. The dimension of the null space at the solution ranged from zero to the number of variables. The condition numbers of the Hessian and the constraints varied from 10^1 to 10^{12}. The definiteness of the problems ranged from being positive definite to almost negative definite.) Using this group, each of the comparisons in the next two sections is based on varying *one* parameter or characteristic at a time, while the others remain unchanged.

Table 3 gives the important characteristics of the group of twenty test problems. By designating each problem by a number, referencing the problem in other tables is greatly simplified. The number of variables is denoted by N. The number of constraints is denoted by NCLIN. The dimension of the null space at the solution is

denoted by NZ. The exponent of the condition number of the constraints and of the objective function is denoted by COND. Finally, the number of negative eigenvalues of the Hessian is denoted by NEGEV.

Table 3

Test problem characteristics

Problem	N	NCLIN	NZ	COND	NEGEV
1	10	50	0	7	0
2	10	50	0	7	0
3	10	50	0	7	0
4	10	50	8	7	0
5	20	50	15	7	0
6	50	20	0	3	0
7	20	20	5	3	0
8	20	20	12	1	0
9	20	20	20	3	0
10	10	50	10	3	0
11	30	30	15	3	0
12	50	50	25	3	0
13	10	50	0	7	2
14	10	50	0	7	8
15	20	50	0	3	3
16	20	50	0	3	10
17	10	50	15	3	2
18	10	50	15	7	5
19	20	50	10	7	2
20	20	50	10	3	10

5.2. Comparison of *QPSFA* constraint deletion strategies

Three parameters involving constraint deletion strategies must be input to *QPSFA* in order to specify the algorithm completely. The computational results obtained by varying each of these parameters will now be compared.

The first parameter is the logical variable PD. If the QP is known to be positive definite, PD is set to true. In this case, *QPSFA* will allow constraints to be deleted from the working set, even though constraints in the working set are violated (see Section 4.6.3). If it is not known that the QP is positive definite, PD must be set to false. In this case, *QPSFA* will allow constraints to be deleted from the working set only if all of the constraints in the working set are satisfied.

Setting PD to true resulted in 0–60% fewer solves of the augmented system than with PD set to false (see Table 4). Note that in this case, the SR should be required less often since constraints with negative multipliers can be deleted even though constraints in the working set are violated. Thus, the only situation that would require the use of the SR is when the current working set defines a vertex, all multipliers are positive, and the current iterate is still infeasible. This situation occurred only a few times in all of the test runs made. Apparently, the strategy of deleting constraints to keep the dimension of the null space larger—and hence allow the Hessian to influence the determination of the search direction more is effective. Thus, PD should be set to true whenever possible.

The second parameter is the real variable THETA, corresponding to θ of (4.46). The values of THETA tested primarily were zero and 10^6. If THETA equals zero, the projected gradient must be zero before constraint deletions are allowed. If THETA equals 10^6, constraint deletions are allowed before the minimum on the manifold is achieved (see Section 4.6.2). When THETA was set to values larger than 10^6, the algorithm found the identical sequence of iterates as with THETA set to 10^6, in

almost all cases. Each problem that was tested with values between zero and 10^6, tended to have a threshold value (or small range of values). Below this threshold, the sequence of iterates found was the same as with THETA equal to zero. Above this threshold, the sequence was the same as with THETA equal to 10^6.

In positive-definite problems, the best value of THETA is strongly correlated to the dimension of the null space at the solution. If NZ is near zero, setting THETA equal to zero resulted in 0–50% fewer solves of the augmented system than with THETA equal to 10^6 (see Table 4). In this case, the solution is (almost) a vertex, so maintaining a larger null space by deleting constraints when not at a minimum resulted in more solves. If NZ was greater than N/4, setting THETA equal to 10^6 resulted in 0–70% fewer solves of the augmented system than with THETA equal to zero (see Table 4). In this case, the solution has a large null space, so maintaining a larger nullspace by deleting constraints was very effective.

In indefinite QP's, the best value of THETA is correlated with the dimension of the null space at the solution in the same way, but not as strongly. This is due to the fact the *QPSFA* cannot delete constraints (in an indefinite QP) when the current iterate violates any constraint in the working set. Thus, *QPSFA* tends to find working sets that define a much smaller dimensional null space when solving indefinite QP's than when solving positive definite QP's.

In LP's, the null space at the solution can always be viewed as having zero dimension. In these problems, a THETA value of zero resulted in slightly fewer solves than with THETA set to 10^6.

Thus, the best setting of THETA depends on the dimension of the null space at the solution. If the dimension is near zero, set THETA to zero. If the dimension is greater than N/4, set THETA to 10^6.

Table 4

Number of solves of the augmented system

when *QPSFA* parameters are varied

Problem	N	NZ	PD / THETA values			
			F/0	F/10^6	T/0	T/10^6
1	10	0	25	33	18	34
2	10	0	31	31	20	30
3	10	0	25	25	19	18
4	10	8	19	19	15	15
5	20	15	35	33	37	18
6	50	0	40	34	35	38
7	20	5	65	57	42	18
8	20	12	30	16	30	10
9	20	20	7	7	7	5
10	10	10	3	2	3	2

The third parameter is the integer variable ISR. The value of ISR specifies which strategy the SR will use to select the constraint to free, when there are two or more suitable constraints to free (see Section 4.6.4). A constraint is suitable if the corresponding β is positive. If ISR equals one, the suitable constraint with the smallest β is freed. If ISR equals two, the suitable constraint with the most negative multiplier is freed. If ISR equals three, the suitable constraint with the smallest ratio of multiplier to corresponding β is freed.

In the problems tested, both positive definite and indefinite, using the smallest β resulted in 0–80% fewer solves of the augmented system than either of the other two strategies. (There were only slight differences in the number of solves between the second and third strategies.) It is not completely clear why the strategies using multiplier information consistently perform less efficiently than the first strategy. Part of the reason is that the multipliers used in the test correspond to the current working set. When the SR is used, a constraint exchange occurs and determines the new working set. The multipliers for this new working set are not readily available.

A summary of the best values for the three parameters is given by the following table.

Table 5

Summary of the best *QPSFA* parameter settings

QP characteristics		Parameter settings		
Hessian	NZ	PD	THETA	ISR
positive definite	$> N/4$	True	10^6	1
positive definite	≈ 0	True	zero	1
null matrix (LP)	zero	False	zero	1
indefinite	$> N/4$	False	10^6	1
indefinite	≈ 0	False	zero	1

5.3. Comparison of *QPSFA* and *QPSOL*

This section will compare the performances of *QPSOL* and *QPSFA* on identical problems. The *QPSFA* parameter values used in the comparison are given in Table 3. The strategy used by the SR (ISR setting) is the same in all cases. However, the values of PD and THETA vary with different characteristics of the QP. Sometimes these characteristics will not be known in advance. However, in many SQP codes the QP's generated are known to be positive definite, so PD can be set true. In order to give THETA a value in an SQP, the dimension of the null space at the solution of the previous QP can be used.

QPSFA has more lattitude than *QPSOL* in the way it selects the initial working set. The choice of initial working set used in the comparison is discussed in Section 5.3.1. The next two sections give results of the comparison for positive definite QP's and indefinite QP's, respectively.

5.3.1. Initial working set selection. Given a starting point, *QPSOL* determines the initial working set from those constraints that are within a small tolerance

of being exactly satisfied. From the same starting point, *QPSFA* can use these almost satisfied constraints, but it could also add violated (by more than the small tolerance) general constraints to the working set. In the comparison, *QPSFA does not* add violated general constraints. These constraints are not added to the initial working set for several reasons. First, different initial working sets would make the comparison more difficult. Second, the violated general constraints that *QPSFA* could add would be added without reference to gradient information. Constraint additions within the algorithm are made using this information. Thus, adding them might increase the number of solves of the augmented system. Third, if NZ is large, decreasing the dimension of the initial null space by adding violated general constraints might also increase the number of solves. Finally, when test problems were run in which *QPSFA did* add violated general constraints to the initial working set, there was no significant change in the number of solves of the augmented system. (If NZ is small, adding as many violated general constraints as possible would appear to be a sensible strategy. However, when this strategy was used, there was still no significant change in the number of solves of the augmented system.)

5.3.2. Positive definite QP's. When *QPSFA* and *QPSOL* were used on positive definite QP's with a wide range of characteristics, there was usually at most a 20% difference and no discernible pattern in the number of solves of the augmented system. (Refer to problems 1, 2, 3, and 6 of Table 6.) However, in two specific cases this was not true. These involve the dimension of the null space at the solution and the conditioning of the problem.

The first case involves QP's which have a value of NZ of approximate dimension N/2. Most QP's tested had a value of N of 20 or less. In these problems, *QPSFA* required about 50% fewer solves than *QPSOL*. (Refer to problems 4, 5, and 7 through 10 of Table 6.) In the QP's tested with values of N from 30 to 50, *QPSFA*

required about 70% fewer solves than *QPSOL*. (Refer to problems 11 and 12 of Table 6.) *QPSOL* tended to find a vertex in the feasibility phase. Therefore, in the optimality phase, constraints had to be deleted to increase the dimension of the null space back up to N/2. On the other hand, *QPSFA* did not find a vertex, since it can delete constraints when constraints in the working set are violated. Thus, the dimension of the null space stayed approximately constant during the first iterations, and then changed *towards* the final value in the later iterations. In problems with larger values of N, we would expect the percentage of fewer solves to continue to increase. (The Appendix contains output from *QPSOL* and *QPSFA* on a positive-definite QP with N = 20 and NZ = 5.)

Table 6

Comparison of the number of solves of the
augmented system in *QPSFA* and *QPSOL*
(positive-definite Hessians)

Problem	N	NZ	*QPSFA*	*QPSOL*
1	10	0	18	20
2	10	0	20	19
3	10	0	19	15
4	10	8	15	16
5	20	15	18	30
6	50	0	35	56
7	20	5	18	53
8	20	12	10	35
9	20	20	5	8
10	10	10	2	5
11	30	15	17	78
12	50	25	38	134

The second case where the performances of *QPSOL* and *QPSFA* differ significantly is when the condition numbers of the Hessian and the constraint matrix decrease. As the condition numbers varied, the number of solves of the augmented sytem required by *QPSOL* did not change significantly or present a discernible pat-

tern. However, as the condition numbers decreased from 10^7 to 10^2, there was a 30% reduction in the number of solves required by *QPSFA* (see Table 7).

This reduction can perhaps be explained by considering the extreme case where the condition numbers are one (i.e., the Hessian is the identity matrix and the only constraints are bounds). In this case, a negative multiplier for a bound *in any working set* implies that the bound is not active at the solution. This suggests that as the condition numbers decrease, the constraints deleted by *QPSFA* are more likely not to be active at the solution and not to be added again in later iterations. On the other hand, *QPSOL* does not delete constraints on the basis of multiplier information until the current iterate is feasible and at a manifold minimum.

Table 7

Comparison of the number of solves of the
augmented system when the condition numbers change
(positive-definite Hessians)

Problem	COND	*QPSFA*	*QPSOL*
1	7	34	30
1	2	18	32
2	7	32	34
2	2	28	26
3	7	34	30
3	2	26	30

The starting point of most of the tests was randomly chosen. However, test runs were made to investigate the affect of different starting points. The different schemes used to determine a starting point included starting at: (*i*) $\bar{x}$ (the optimal solution) plus a step in the direction of $g(\bar{x})$ (the gradient at $\bar{x}$), (*ii*) $\bar{x}$ plus a step opposite $g(\bar{x})$, (*iii*) the middle of the region defined by the upper and lower bounds of the variables, and (*iv*) the minimum of the QP excluding the general constraints. In each of these cases, the relative performance (in terms of number of solves of the

augmented system) of *QPSOL* and *QPSFA* did not change significantly. However, if we compare the objective values at the starting point, the first feasible point, and the optimal point, *QPSFA* and *QPSOL* differ greatly in case (*iv*). In the runs made (see Table 8), the initial objective value was of order -10^5, and the optimal objective value was of order -10^4. In each case, the first feasible point found by *QPSFA* was the optimal point. In comparison, the first feasible point found by *QPSOL* had an objective value of approximately order 10^5. Thus, by ignoring the Hessian information during the feasible phase, *QPSOL* found an initial feasible point with an objective value significantly greater than both the objective value at the initial point and the optimal objective point.

Table 8

Comparison of objective values using the
unconstrained minimum as the starting point

N	NZ	objective value at			
		initial point	first feasible point—*QPSOL*	first feasible point—*QPSFA*	optimal point
10	0	-5.9+05	1.1+05	-1.7+04	-1.7+04
10	0	-3.0+05	3.7+04	-1.7+04	-1.7+04
10	5	-2.3+05	1.5+07	-1.6+04	-1.6+04
10	0	-6.0+05	4.9+03	-2.1+04	-2.1+04

5.3.3. Indefinite QP's. When *QPSOL* and *QPSFA* were used on indefinite QP's with a wide range of characteristics, *QPSFA* required from −20 to +100% more solves of the augmented system than *QPSOL* (see Table 9). Test problems with a value of NZ near N/2 or with small condition numbers did not yield significantly different numbers of solves, as they did in positive definite QP's. This was expected since *QPSFA* (when solving an indefinite QP) does not delete constraints from the working set if any constraint in the working set is violated. Varying the starting

point did not result in any significant change in the relative performance of *QPSOL* and *QPSFA*. (In many instances, no comparison was possible since different local minima were found.)

Table 9

Comparison of the number of solves of the
augmented system in *QPSFA* and *QPSOL*
(indefinite Hessians)

Problem	N	NZ	*QPSFA*	*QPSOL*
13	10	0	46	40
14	10	0	26	16
15	20	0	108	82
16	20	0	49	47
17	10	15	32	22
18	10	15	34	32
19	20	10	76	84
20	20	10	71	46

Even though *QPSFA* takes more solves of the augmented system than does *QPSOL* (when solving indefinite problems), *QPSFA* may still require less computation, particularly in a large-scale problem. This will depend on what portion of the total work is needed to form the factorizations of the initial augmented system. If the logic of *QPSOL* were adapted to a large-scale problem, it would require that one additional factorization of K be performed, since the factorization used in the feasibility phase cannot be used in the optimality phase. (Here, we are assuming that successive augmented systems are being solved as in Section 3.3.4.) Thus, if this factorization constitutes a large portion of the total work and *QPSFA* calculates this factorization only a few times (perhaps once, but in any event one time fewer than *QPSOL*), *QPSFA* may require less total computational effort, even though it needs more solves of the augmented system.

Bibliography

Avriel, M. (1976). *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Bartels, R. H., Golub, G. H. and Saunders, M. A. (1970). "Numerical techniques in mathematical programming", in *Nonlinear Programming* (J. B. Rosen, O. L. Mangasarian and K. Ritter, eds.), pp. 123–176, Academic Press, London and New York.

Best, M. J. (1984). Equivalence of some quadratic programming algorithms, *Math. Prog.* **30**, pp. 71–87.

Bisschop, J. and Meeraus, A. (1977). Matrix augmentation and partitioning in the updating of the basis inverse, *Math. Prog.* **13**, pp. 241–254.

Bisschop, J. and Meeraus, A. (1980). Matrix augmentation and structure preservation in linearly constrained control problems, *Math. Prog.* **18**, pp. 7–15.

Bunch, J. R. and Kaufman, L. C. (1977). Some stable methods for calculating inertia and solving symmetric linear equations, *Mathematics of Computation* **31**, pp. 163–179.

Bunch, J. R. and Kaufman, L. C. (1980). A computational method for the indefinite quadratic programming problem, *Linear Algebra and its Applics.* **34**, pp. 341–370.

Bunch, J. R. and Parlett, B. N. (1971). Direct methods for solving symmetric indefinite systems of linear equations, *SIAM J. Numer. Anal.* **8**, pp. 639–655.

Conn, A. R. (1976). Linear programming via a non-differentiable penalty function, *SIAM J. Numer. Anal.* **13**, pp. 145–154.

Conn, A. R. and Gould N. I. M. (1984). On the location of directions of infinite descent for nonlinear programming algorithms, *SIAM J. Numer. Anal.* **21**, pp. 1162–1179.

Conn, A. R. and Sinclair, J. W. (1975). Quadratic programming via a non-differentiable penalty function, Report 75/15, Department of Combinatorics and Optimization, University of Waterloo, Canada.

Cottle, R. W. (1974). Manifestations of the Schur complement, *Linear Algebra and its Applics.* **8**, pp. 189–211.

Cottle, R. W. (1979). "Fundamentals of quadratic programming and linear complementarity", in *Engineering Plasticity by Mathematical Programming* (M. Z. Cohn and G. Maier, eds.), pp. 293–323, Pergamon Press, New York.

Cox, M. G. (1981). The least squares solution of overdetermined linear equations having band or augmented band structures, *IMA J. Numer. Anal.* **1**, pp. 3–22.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

Djang, A. (1980). *Algorithmic Equivalence in Quadratic Programming*, Ph.D. Thesis, Stanford University, California.

Duff, I. S. and Reid, J. K. (1984). Direct methods for solving sparse systems of linear equations, *SIAM J. Sci. Stat. Comput.* **5**, pp. 605–619.

Fletcher, R. (1973). An exact penalty function for nonlinear programming with inequalities, *Math. Prog.* **5**, pp. 129–150.

Fletcher, R. (1980). *Practical Methods of Optimisation, Volume 1, Unconstrained Optimisation*, John Wiley & Sons, New York and Toronto.

Fletcher, R. (1981). *Practical Methods of Optimisation, Volume 2, Constrained Optimisation*, John Wiley & Sons, New York and Toronto.

George, A. and Liu, J. W. H. (1980). A minimal storage implementation of the minimum degree algorithm, *SIAM J. Numer. Anal.* **17**, pp. 282–299.

Gill, P. E., Gould, N. I. M., Murray, W., Saunders, M. A. and Wright, M. H. (1982a). Range-space methods for convex quadratic programming, Report SOL 82-14, Department of Operations Research, Stanford University, California.

Gill, P. E., Gould, N. I. M., Murray, W., Saunders, M. A. and Wright, M. H. (1982b). A range-space method for quadratic programming problems with bounds and general constraints, Report SOL 82-15, Department of Operations Research, Stanford University, California.

Gill, P. E. and Murray, W. (eds.) (1974). *Numerical Methods for Constrained Optimization*, Academic Press, London and New York.

Gill, P. E. and Murray, W. (1977). "Linearly constrained problems including linear and quadratic programming", in *The State of the Art in Numerical Analysis* (D. Jacobs, ed.), pp. 313–363, Academic Press, London and New York.

Gill, P. E. and Murray, W. (1978a). Numerically stable methods for quadratic programming, *Math. Prog.* **14**, pp. 349–372.

Gill, P. E. and Murray, W. (1978b). The design and implementation of software for unconstrained optimization, in *Design and Implementation of Optimization Software* (H. Greenberg, ed.), pp. 221–234, Sijthoff and Noordhoff, Netherlands.

Gill, P. E. and Murray, W. (1979). The computation of Lagrange multiplier estimates for constrained minimization, *Math. Prog.* **17**, pp. 32–60.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1980). Methods for large-scale nonlinear optimization, Report SOL 80-8, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1981). QP-based methods for large-scale nonlinearly constrained optimization, Report SOL 81-1, Department of Operations Research, Stanford University, California. To appear in *Nonlinear Programming 4*, (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), Academic Press, London and New York.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1982). Procedures for optimization problems with a mixture of bounds and general linear constraints, Report SOL 82-6, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1983a). User's guide for SOL/QPSOL: A Fortran package for quadratic programming, Report SOL 83-7, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1983b). On the representation of a basis for the null space, Report SOL 83-19, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984a). Sparse matrix methods in optimization, *SIAM J. Sci. Stat. Comput.* **5**, pp. 562–589.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984b). Software and its relationship to methods, Report SOL 84-10, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1985a). Model building and practical aspects of nonlinear programming, Report SOL 85-2, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1985b). Large-scale quadratic programming using the Schur complement update, Report SOL to appear, Department of Operations Research, Stanford University, California.

Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London and New York.

Goldfarb, D. (1972). "Extensions of Newton's method and simplex methods for solving quadratic programs", in *Numerical Methods for Non-linear Optimisation* (F. A. Lootsma, ed.), pp. 239–254, Academic Press, London and New York.

Goldfarb, D. and Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs, *Math. Prog.* **27**, pp. 1–33.

Gould, N. I. M. (1982). *Numerical Methods for Linear and Quadratic Programs*, D. Phil. Thesis, Oxford University, England.

Gould, N. I. M. (1984). On the location of directions of infinite descent for nonlinear programming algorithms, *SIAM J. Numer. Anal.* **21**, pp. 1162–1179.

Han, S.-P. (1981). "Solving quadratic programs by an exact-penalty function", in *Nonlinear Programming 4*, (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), pp. 25–55, Academic Press, London and New York.

Knuth, D. E. (1979). *TEX and METAFONT, New Directions in Typesetting*, American Mathematical Society and Digital Press, Bedford, Massachusetts.

Murray, W. (1971). An algorithm for finding a local minimum of an indefinite quadratic program, Report NAC 1, National Physical Laboratory, England.

Murray, W. and Wright, M. H. (1980). Computation of the search direction in constrained optimization algorithms, Report SOL 80-2, Department of Operations Research, Stanford University, to appear in *Math. Prog. Study on Constrained Optimisation.*

Powell. M. J. D. (1981). An upper-triangular matrix method for quadratic programming, in *Nonlinear Programming 4*, (O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds.), pp. 1–24, Academic Press, London and New York.

Stewart, G. W. (1973). *Introduction to Matrix Computations*, Academic Press, London and New York.

Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem*, Oxford University Press.

Appendix

This appendix describes the output from *QPSOL* and *QPSFA* on a positive-definite QP with 20 variables (N = 20), and 20 general constraints. The QP has five bounds and ten general constraints in the active set. The dimension of the null space at the solution is therefore five (NZ = 5).

Table 10*a* lists the output of the feasibility phase of *QPSOL*. Table 10*b* lists the output of the optimality phase of *QPSOL*. Table 11 lists the output of *QPSFA*.

The column headings of interest are now described. "ITN" gives the iteration number. An iteration includes at most one deletion and one addition. The index of the constraint deleted from or added to the working set is given by "JDEL" and "JADD", respectively. "STEP" gives the step length α, taken along the search direction. The value of the quadratic objective function is given by "OBJECTIVE". "NCOLZ" gives the number of columns of the null space. The Euclidean norm of the projected gradient is given by "NORM ZTG". "COND T" and "COND ZHZ" are estimates of the condition numbers of the constraints in the working set and the projected Hessian. The number of general constraints violated by the current iterate is given by "NUMINF". "SUMINF" gives the sum of the residuals of all violated constraints. "NORM RES" gives the Euclidean norm of the residuals of all violated constraints *in the working set*. If the Singularity Rule is used due to singularity in the constraints, a "T" would appear in the column "SINGT". A single line of output is printed each time a constraint is deleted or added, or a unit step is taken.

Comparing the output will highlight some of the important features of *QPSFA* and differences between *QPSFA* and *QPSOL*. The step in *QPSOL* when a constraint is added is always nonzero. In *QPSFA*, when a violated constraint is added, the

step length is zero. Such a zero step occurs five times in Table 11. The objective value always decreases in the optimality phase of *QPSOL*. In *QPSFA* the objective value increases during certain iterations. This occurs because *QPSFA* does not necessarily find a descent direction (when the current iterate is not feasible). A feasible vertex is determined by the feasibility phase of *QPSOL* even though the initial point has a null space of dimension nine. The optimality phase of *QPSOL* starts at this feasible vertex and must build the dimension of the null space back up to five. *QPSFA* leaves NCOLZ at nine or ten, until the final iterations in which the correct null space of dimension five is found. The norm of the projected gradient must be close to zero before *QPSOL* allows a constraint deletion. Since *QPSFA* allows constraint deletions before the norm of the projected gradient vanishes, this norm is close to zero only on the last iteration. A feasible point (NUMINF = 0) is found by *QPSFA* on the last iteration. (This occured in approximately 20% of the test runs.) The sum of the infeasibilities is decreasing in both the feasibility phase of *QPSOL* and in *QPSFA*. (This decrease is used to show convergence of these algorithms.) The Singularity Rule was not needed by *QPSFA* for this problem. It was needed only several times during all of the tests with positive-definite Hessians when constraint deletions were allowed before the current iterate satisfied the working set. In indefinite problems, the Singularity Rule was used more frequently.

ITN	JDEL	JADD	STEP	COND T	NUMINF	SUMINF
0	0	0	0.00D-01	1.00D 00	9	2.260184D 04
1	0	22L	5.54D 02	1.00D 00	8	2.035363D 04
2	0	16U	4.19D 03	1.00D 00	6	7.914267D 03
3	0	20L	5.96D 03	1.00D 00	5	3.413305D 03
4	0	18U	1.74D 03	1.00D 00	5	3.014448D 03
5	0	12L	3.31D 03	1.00D 00	5	2.778220D 03
6	0	4L	1.99D 03	1.00D 00	5	2.646047D 03
7	0	14L	2.59D 04	1.00D 00	5	1.767774D 03
8	0	21U	5.88D 03	1.73D 00	5	1.649971D 03
9	0	8L	1.83D 05	2.22D 00	5	1.629494D 03
10	3L	3U	3.22D 03	2.22D 00	5	9.044010D 02
11	22L	22U	2.52D 03	1.17D 00	5	7.989301D 02
12	16U	6L	5.28D 04	3.13D 00	4	6.861067D 02
13	5U	5L	7.33D 03	3.13D 00	3	3.134075D 02
14	4L	33L	4.06D 04	5.27D 00	3	2.699737D 02
15	19U	24U	9.70D 04	1.28D 01	2	2.003507D 02
16	3U	25U	3.16D 03	3.83D 01	2	1.984985D 02
17	6L	26U	1.10D 05	3.51D 01	2	1.274191D 02
18	25U	16U	2.16D 04	1.80D 01	2	1.261151D 02
19	13U	27L	1.77D 05	1.65D 02	1	7.581578D 01
20	1U	1L	1.61D 04	1.65D 02	1	7.158890D 01
21	16U	29L	1.47D 04	4.20D 02	1	6.885000D 01
22	27L	25U	1.49D 05	1.06D 02	1	4.170446D 01
23	8L	28U	7.31D 05	2.24D 02	1	2.230162D 01
24	9L	13L	3.09D 04	1.58D 02	1	2.103163D 01
25	28U	3L	1.84D 06	7.35D 02	1	2.062488D 01
26	5L	27L	2.35D 06	1.06D 03	1	1.890328D 01
27	3L	28U	3.32D 06	5.29D 02	1	1.614730D 01
28	7L	16U	3.69D 05	1.31D 02	1	1.139880D 01
29	28U	30L	3.05D 06	2.20D 02	1	8.522523D 00
30	11U	23U	3.26D 05	1.92D 02	0	0.000000D-01

TABLE 10*a*

Solution of a QP problem using *QPSOL*

Feasibility phase

ITN	JDEL	JADD	STEP	NHESS	OBJECTIVE	NCOLZ	NORM GFREE	NORM ZTG	COND T	COND ZHZ
0	0	0	0.00D-01	1	1.3018D 07	0	2.77D 03	0.00D-01	1.9D 02	1.0D 00
0	21U	0	0.00D-01	2	1.3018D 07	1	2.77D 03	1.64D 03	2.0D 02	1.0D 00
1	0	28U	6.28D-01	3	1.0284D 07	0	1.90D 03	0.00D-01	2.0D 02	1.0D 00
1	33L	0	0.00D-01	4	1.0284D 07	1	1.90D 03	8.46D 02	8.5D 01	1.0D 00
2	0	19L	8.32D-01	5	8.1010D 06	0	1.81D 03	0.00D-01	8.4D 01	1.0D 00
2	27L	0	0.00D-01	6	8.1010D 06	1	1.81D 03	4.77D 02	8.3D 01	1.0D 00
3	0	0	1.00D 00	7	7.5505D 06	1	1.41D 03	5.68D-14	8.3D 01	1.0D 00
3	1L	0	0.00D-01	8	7.5505D 06	2	1.62D 03	1.54D 03	9.2D 01	1.5D 00
4	0	1U	3.96D-02	9	7.2483D 06	1	1.35D 03	5.09D-14	8.3D 01	1.0D 00
4	10U	0	0.00D-01	10	7.2483D 06	2	2.29D 03	1.17D 03	7.7D 01	3.4D 00
5	0	7U	1.50D-01	11	6.9807D 06	1	2.16D 03	9.60D 02	7.7D 01	1.0D 00
6	0	0	1.00D 00	12	6.5082D 06	1	1.90D 03	0.00D-01	7.7D 01	1.0D 00
6	29L	0	0.00D-01	13	6.5082D 06	2	1.90D 03	3.39D 01	7.0D 01	2.0D 00
7	0	0	1.00D 00	14	6.5070D 06	2	1.90D 03	1.14D-14	7.0D 01	2.0D 00
7	13L	0	0.00D-01	15	6.5070D 06	3	2.22D 03	1.63D 03	6.9D 01	3.0D 00
8	0	29L	2.90D-01	16	4.4595D 06	2	1.54D 03	1.16D 03	1.4D 02	3.0D 00
9	0	0	1.00D 00	17	2.3887D 06	2	2.77D 02	3.50D-13	____ 02	3.0D 00
9	30L	0	0.00D-01	18	2.3887D 06	3	2.77D 02	1.70D 02	9.6D 01	3.0D 00
10	0	0	1.00D 00	19	2.3504D 06	3	2.46D 02	5.63D-13	9.6D 01	3.0D 00
10	29L	0	0.00D-01	20	2.3504D 06	4	2.46D 02	1.26D 02	2.0D 01	3.8D 00
11	0	0	1.00D 00	21	2.3193D 06	4	2.23D 02	8.88D-13	2.0D 01	3.8D 00
11	26U	0	0.00D-01	22	2.3193D 06	5	2.23D 02	5.51D 01	5.4D 00	1.7D 01
12	0	21U	4.63D-02	23	2.3170D 06	4	2.22D 02	1.45D 01	7.2D 00	3.9D 00
13	0	0	1.00D 00	24	2.3164D 06	4	2.19D 02	1.47D-12	7.2D 00	3.9D 00
13	12L	0	0.00D-01	25	2.3164D 06	5	8.59D 02	7.73D 02	7.2D 00	4.3D 00
14	0	0	1.00D 00	26	5.9023D 05	5	3.51D 02	2.54D-12	7.2D 00	4.3D 00
14	7U	0	0.00D-01	27	5.9023D 05	6	4.99D 02	4.00D 02	7.2D 00	7.5D 00
15	0	26U	2.82D-02	28	5.4542D 05	5	4.90D 02	3.70D 02	5.1D 01	7.3D 00
16	0	29L	6.28D-01	29	2.6169D 04	4	3.68D 02	1.31D 02	1.2D 02	6.2D 00
17	0	30L	4.62D-01	30	-1.3270D 04	3	3.65D 02	5.07D 01	1.6D 02	3.1D 00
18	0	0	1.00D 00	31	-1.5912D 04	3	3.64D 02	8.68D-13	1.6D 02	3.1D 00
18	1U	0	0.00D-01	32	-1.5912D 04	4	3.65D 02	5.51D 01	1.7D 02	1.0D 01
19	0	1L	2.11D-01	33	-2.6801D 04	3	3.61D 02	8.93D-13	1.6D 02	3.1D 00
19	15U	0	0.00D-01	34	-2.6801D 04	4	3.89D 02	2.54D 01	1.6D 02	3.1D 01
20	0	27L	6.38D-01	35	-4.3425D 04	3	3.86D 02	9.07D 00	1.6D 02	3.4D 01
21	0	0	1.00D 00	36	-4.5899D 04	3	3.84D 02	1.46D-12	1.6D 02	3.4D 01
21	14L	0	0.00D-01	37	-4.5899D 04	4	4.14D 02	1.40D 01	1.6D 02	1.1D 02
22	0	0	1.00D 00	38	-6.4097D 04	4	4.14D 02	2.02D-12	1.6D 02	1.1D 02
22	1L	0	0.00D-01	39	-6.4097D 04	5	4.14D 02	2.74D-01	1.7D 02	1.4D 02
23	0	0	1.00D 00	40	-6.4107D 04	5	4.14D 02	2.89D-12	1.7D 02	1.4D 02

TABLE 10*b*

Solution of a QP problem using *QPSOL*

Optimality phase

ITN	JDEL	JADD	STEP	IBIESS	OBJECTIVE	NCOLZ	NORM GFREE	NORM ZTG	COND T	COND ZHZ	NUMINF	NORM RES	SUM INF	STNGT
0	0	0	0.00D-01	10	5.4554D 07	9	9.81D 03	9.81D 03	1.0D 00	2.7D 02	9	0.00D-01	2.3D 04	F
0	9L	0	0.00D-01	11	5.4554D 07	10	9.95D 03	9.95D 03	1.0D 00	2.7D 02	9	0.00D-01	2.3D 04	F
1	0	26U	0.00D-01	12	5.4554D 07	9	9.95D 03	7.47D 03	1.0D 00	4.0D 01	9	3.22D 03	2.3D 04	F
1	11U	0	0.00D-01	13	5.4554D 07	10	9.95D 03	7.67D 03	1.0D 00	4.0D 01	9	3.22D 03	2.3D 04	F
2	0	22U	0.00D-01	14	5.4554D 07	9	9.95D 03	5.20D 03	1.3D 00	4.8D 01	9	3.22D 03	2.3D 04	F
2	10U	0	0.00D-01	15	5.4554D 07	10	9.95D 03	5.76D 03	1.3D 00	4.8D 01	9	3.22D 03	2.3D 04	F
3	0	20L	3.71D-01	16	1.9542D 07	9	5.74D 03	3.49D 03	1.4D 00	5.3D 01	9	2.02D 03	8.7D 03	F
3	17L	0	0.00D-01	17	1.9542D 07	10	5.76D 03	3.64D 03	1.4D 00	5.3D 01	9	2.02D 03	8.7D 03	F
4	0	33L	0.00D-01	18	1.9542D 07	9	5.76D 03	3.13D 03	2.4D 00	4.4D 01	9	2.15D 03	8.7D 03	F
4	19U	0	0.00D-01	19	1.9542D 07	10	6.27D 03	4.16D 03	2.6D 00	4.4D 01	9	2.15D 03	8.7D 03	F
5	0	17L	0.00D-01	20	1.9542D 07	9	6.25D 03	4.11D 03	2.1D 00	4.4D 01	9	2.15D 03	8.7D 03	F
5	33L	0	0.00D-01	21	1.9542D 07	10	6.25D 03	4.13D 03	1.4D 00	4.4D 01	9	2.15D 03	8.7D 03	F
6	0	4L	3.72D-01	22	6.3342D 06	9	3.81D 03	2.50D 03	1.4D 00	2.3D 01	5	1.27D 03	3.4D 03	F
6	5U	0	0.00D-01	23	6.3342D 06	10	3.81D 03	2.61D 03	1.4D 00	2.5D 01	5	1.27D 03	3.4D 03	F
7	0	18U	7.51D-01	24	-8.1781D 05	9	7.51D 02	6.49D 02	1.4D 00	3.0D 01	5	3.17D 02	7.7D 02	F
7	13U	0	0.00D-01	25	-8.1781D 05	10	7.54D 02	6.64D 02	1.3D 00	3.0D 01	5	3.17D 02	7.7D 02	F
8	0	21U	4.07D-01	26	-9.9624D 05	9	4.08D 02	2.46D 02	3.1D 00	2.5D 01	4	1.88D 02	4.3D 02	F
8	3L	0	0.00D-01	27	-9.9624D 05	10	4.20D 02	3.38D 02	1.3D 00	2.5D 01	4	1.88D 02	4.3D 02	F
9	0	23U	1.39D-02	28	-9.9673D 05	9	4.15D 02	2.95D 02	8.5D 00	2.3D 01	4	1.85D 02	4.2D 02	F
9	4L	0	0.00D-01	29	-9.9673D 05	10	4.16D 02	3.08D 02	6.8D 00	2.9D 01	4	1.85D 02	4.2D 02	F
10	0	25U	0.00D-01	30	-9.9673D 05	9	4.16D 02	3.01D 02	9.7D 00	2.3D 01	4	1.96D 02	4.2D 02	F
10	15U	0	0.00D-01	31	-9.9673D 05	10	4.29D 02	3.03D 02	9.7D 00	1.4D 02	4	1.96D 02	4.2D 02	F
11	0	19L	1.94D-01	32	-9.1302D 05	9	3.46D 02	1.89D 02	9.7D 00	1.3D 02	4	1.58D 02	3.4D 02	F
12	0	16U	5.71D-01	33	-5.5941D 05	8	3.24D 02	3.96D 01	7.4D 00	1.3D 02	4	6.77D 01	1.6D 02	F
13	0	24U	6.70D-01	34	-3.2612D 05	7	3.30D 02	1.29D 01	8.6D 00	9.7D 01	4	2.23D 01	6.3D 01	F
14	0	27L	4.43D-01	35	-2.6235D 05	6	3.37D 02	6.55D 00	1.6D 02	8.0D 01	4	1.25D 01	3.2D 01	F
14	7L	0	0.00D-01	36	-2.6235D 05	7	3.41D 02	1.72D 01	2.6D 01	8.0D 01	4	1.25D 01	3.2D 01	F
15	0	29L	8.57D-01	37	-1.1620D 05	6	3.71D 02	6.71D-01	6.6D 01	1.4D 02	3	1.78D 00	2.3D 00	F
15	1U	0	0.00D-01	38	-1.1620D 05	7	3.71D 02	6.73D-01	8.0D 01	1.6D 02	3	1.78D 00	2.3D 00	F
16	0	28U	9.21D-01	39	-7.1316D 04	6	4.03D 02	4.39D-02	8.0D 01	1.7D 02	3	1.41D-01	1.8D-01	F
17	0	30L	1.41D-01	40	-7.0536D 04	5	4.03D 02	3.51D-02	2.2D 02	1.1D 02	3	1.21D-01	1.6D-01	F
18	0	0	1.00D 00	41	-6.4107D 04	5	4.14D 02	2.70D-15	2.2D 02	1.1D 02	0	1.24D-14	0.00D-01	F

TABLE 11

Solution of a QP problem using *QPSFA*

END

DTIC

5-86